# CS8381 DATA STRUCTURES LABORATORY

## OBJECTIVES

- To implement linear and non-linear data structures
- To understand the different operations of search trees
- To implement graph traversal algorithms
- To get familiarized to sorting and searching algorithms

## OUTCOMES:

At the end of the course, the students will be able

1. Write functions to implement linear and non-linear data structure operations
2. Suggest appropriate linear / non-linear data structure operations for solving a given problem
3. Appropriately use the linear / non-linear data structure operations for a given problem
4. Apply appropriate hash functions that result in a collision free scenario for data storage and retrieval

## CS8381-DATA STRUCTURES LABORATORY

INDEX


| EX. NO. | NAME OF THE EXPERIMENT | PAGE NO. |
|---|---|---|
| 1. | Array implementation of ADTs | |
| | 1.A Stack ADTs | |
| | 1.B Queue ADTs | |
| 2. | Array implementation of List ADT | |
| 3. | Linked list implementation | |
| | 3.A.Linked list implementation of Stack ADTs | |
| | 3.B.Linked list implementation of Queue ADTs | |
| 4. | Applications of List, Stack and Queue ADTs | |
| | 4.A. Implementation of Polynomial ADT | |
| | 4.B. Conversion of Infix into Postfix Expression | |
| 5. | Implementation of Binary Trees and operations of Binary Trees | |
| 6. | Implementation of Binary Search Trees | |
| 7. | Implementation of AVL Trees | |
| 8. | Implementation of Heaps using Priority Queues. | |
| 9. | 9.A.Graph representation | |
| | 9.B.Graph Traversal-BFS and DFS | |
| 10. | Implementation of Searching algorithms –Linear Search and Binary Search. | |
| | Implementation of Sorting algorithms –Bubble sort,ShellSort,Quick Sort. | |
| 11. | Implementation of Hashing Technique. | |
| **CONTENT BEYOND THE SYLLABUS** | | |
| 1. | Singly Linked List | |
| 2. | Double Ended Queue | |

## EX.NO:1A ARRAY IMPLEMENTATION OF STACK ADT

**AIM:**

Aim is to write a program in C to implement the stack ADT using array concept that performs all the operations of stack.

**DESCRIPTION:**

A stack data structure can be implemented using one dimensional array. But stack implemented using array, can store only fixed number of data values. This implementation is very simple, just define a one dimensional array of specific size and insert or delete the values into that array by using LIFO principle with the help of a variable 'top'. Initially top is set to -1. Whenever we want to insert a value into the stack, increment the top value by one and then insert. Whenever we want to delete a value from the stack, then delete the top value and decrement the top value by one.

**ALGORITHM:**

STEP 1: Define an array to store the element.

STEP 2: Get the users' choice.

STEP 3: If the option is 1 perform creation operation and goto step4.

If the option is 2 perform insertion operation and goto step5.

If the option is 3 perform deletion operation and goto step6.

If the option is 4 perform display operation and goto step7.

STEP 4: Create the stack. Initially get the limit of stack and the get the items. If the limit of stack is exceeds print the message unable to create the stack.

STEP 5: Get the element to be pushed. If top pointer exceeds stack capacity. Print Error message that the stack overflow. If not, increment the top pointer by one and store the element in the position which is denoted by top pointer.

STEP 6: If the stack is empty, then print error message that stack is empty. If not fetch the element from the position which is denoted by top pointer and decrement the top pointer by one

STEP 7: If the top value is not less than the 0 the stack is display otherwise print the message "stack is empty".

STEP 8: Stop the execution.

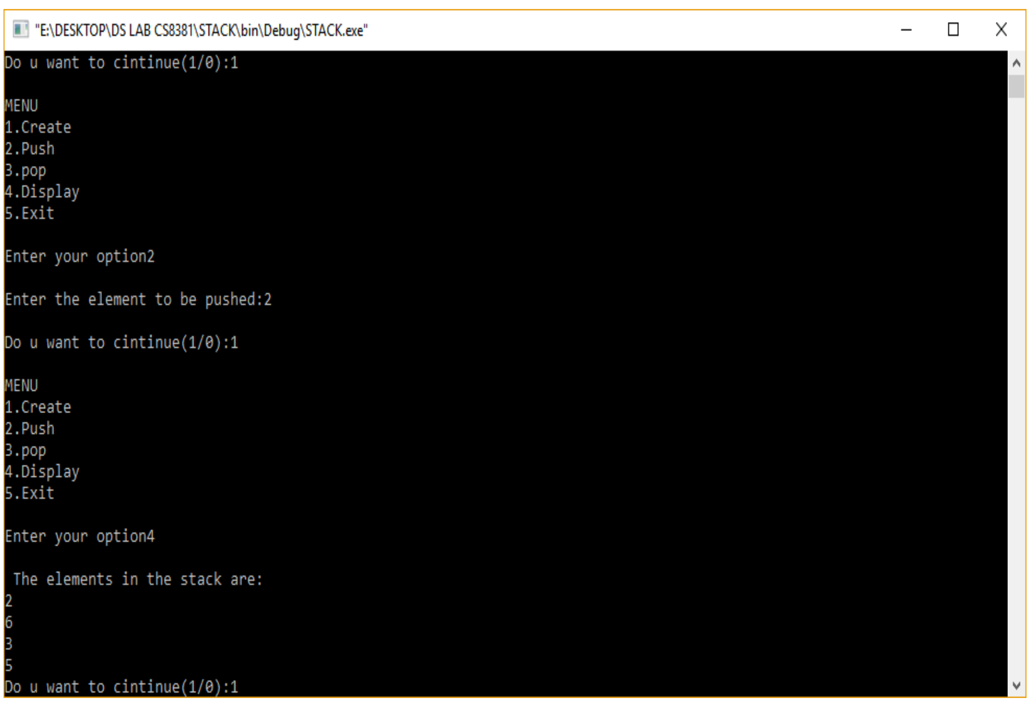
**PROGRAM:**

```
#include<stdio.h>
#include<conio.h>
#define max 20
int opt, a[20],i,top=0,n;
void main()
{
     void create(),push(),pop(),disp();
     int wish;
     do
     {
     clrscr();
     printf("\nMENU");
     printf("\n1.Create\n2.Push\n3.pop\n4.Display\n5.Exit\n");
     printf("\nEnter your option");
     scanf("%d",&opt);
     switch(opt)
     {
          case 1:create();break;
          case 2:push();break;
          case 3:pop();break;
          case 4:disp();break;
          case 5:exit(0);
     }
     printf("\nDo u want to cintinue(1/0):");
     scanf("%d",&wish);
     }while(wish==1);}
void create()
{
     printf("\n Enter the limit of stack");
     scanf("%d",&n);if(n<max)
```

```
{
        printf("\nEnter the items");
        for(i=0;i<n;i++)
        scanf("%d",&a[i]);
        top=n-1;
    }
    else
    printf("\nUnable to create the stack");
}
void push()
{
    int x;
    if(top<max){
            printf("\nEnter the element to be pushed:");
        scanf("%d",&x);
        top=top+1;
        a[top]=x;
        n=top;
    }
    else
    printf("\n Stack is full");
}
void pop()
{
    if(top<0)
    printf("\n Stack is empty");
    else
    {
        printf("\nThe element popped is %d",a[top]);
        top=top-1;
        n=top;
```

```
}}
void disp()
{
    if(top<0)
    printf("\n Stack is empty");
    else
    {
        printf("\n The elements in the stack are:");
        for(i=top;i>=0;i--)
            printf("\n%d",a[i]);
    }
}
```

**OUTPUT:**



```
"E:\DESKTOP\DS LAB CS8381\STACK\bin\Debug\STACK.exe"
Do u want to cintinue(1/0):1

MENU
1.Create
2.Push
3.pop
4.Display
5.Exit

Enter your option2

Enter the element to be pushed:2

Do u want to cintinue(1/0):1

MENU
1.Create
2.Push
3.pop
4.Display
5.Exit

Enter your option4

 The elements in the stack are:
2
6
3
5
Do u want to cintinue(1/0):1
```

```
"E:\DESKTOP\DS LAB CS8381\STACK\bin\Debug\STACK.exe"
1.Create
2.Push
3.pop
4.Display
5.Exit

Enter your option4

 The elements in the stack are:
2
6
3
5
Do u want to cintinue(1/0):1

MENU
1.Create
2.Push
3.pop
4.Display
5.Exit

Enter your option3

The element popped is 2
Do u want to cintinue(1/0):4

Process returned 4 (0x4)   execution time : 349.030 s
Press any key to continue.
```

## RESULT:

Thus a C program for Stack using ADT was implemented successfully.

**EX. NO: 1B** **QUEUE ADT USING ARRAY**

**AIM:**

To write a program for Queue using array implementation.

**DESCRIPTION:**

A queue data structure can be implemented using one dimensional array. But, queue implemented using array can store only fixed number of data values. The implementation of queue data structure using array is very simple, just define a one dimensional array of specific size and insert or delete the values into that array by using FIFO (First In First Out) principle with the help of variables 'front' and 'rear'. Initially both 'front' and 'rear' are set to -1. Whenever, we want to insert a new value into the queue, increment 'rear' value by one and then insert at that position. Whenever we want to delete a value from the queue, then increment 'front' value by one and then display the value at 'front' position as deleted element.

**ALGORITHM:**

1. Define a array which stores queue elements..
2. The operations on the queue are
    a. a)INSERT data into the queue
    b. b)DELETE data out of queue
3. INSERT DATA INTO queue
    a. Enter the data to be inserted into queue.
    b. If TOP is NULL
        i. The input data is the first node in queue.
        ii. The link of the node is NULL.
        iii. TOP points to that node.
    c. If TOP is NOT NULL
        i. The link of TOP points to the new node.
        ii. TOP points to that node.
4. DELETE DATA FROM queue
    a. If TOP is NULL
        i. the queue is empty
    b. If TOP is NOT NULL
        i. The link of TOP is the current TOP.
        ii. The pervious TOP is popped from queue.
5. The queue represented by linked list is traversed to display its content.

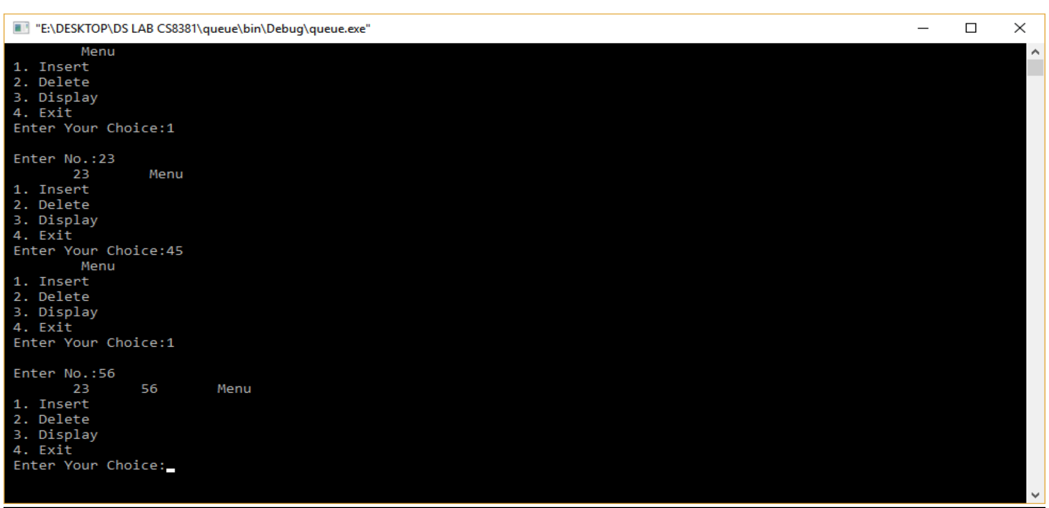
**PROGRAM**

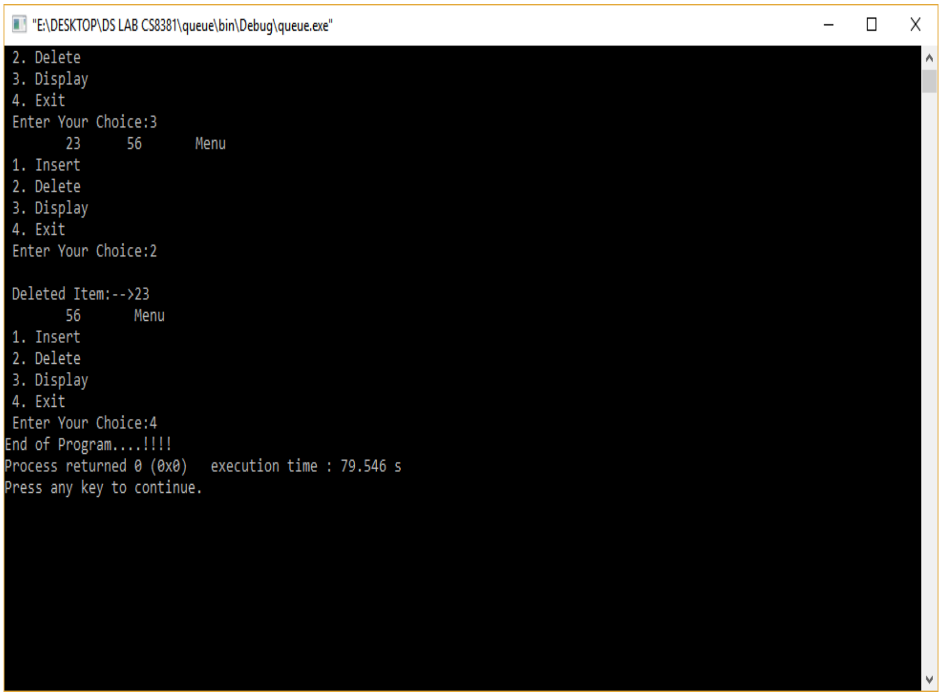
```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define SIZE 5
int front = - 1;
int rear = - 1;
int q[SIZE];
void insert( );
void del( );
void display( );
void main( )
{
int choice;
 do
{
printf("\t Menu");
printf("\n 1. Insert");
printf("\n 2. Delete");
 printf("\n 3. Display ");
printf("\n 4. Exit");
printf("\n Enter Your Choice:");
scanf("%d", &choice);
switch(choice)
{
case 1:
insert( ); display( ); break;
case 2:
del( ); display( ); break;
case 3:display( );
```

```
break;
case 4:
printf("End of Program....!!!!");
exit(0);
}}while(choice != 4);}
void insert( )
{
int no;
printf("\n Enter No.:");
scanf("%d", &no);

if(rear < SIZE - 1)
{
q[++rear]=no;
if(front == -1)
front=0;// front=front+1;
}
else
{
printf("\n Queue overflow");
}}

void del( )
{
if(front == - 1)
{
printf("\n Queue Underflow");
return;
}
else
```

```
{
printf("\n Deleted Item:-->%d\n", q[front]);
}
if(front == rear)
{
front = - 1;
rear = - 1;
 }
else
{front = front + 1;
}}
void display( )
{
int i;
if( front == - 1)
{
printf("\nQueue is empty....");
return;
}
for(i = front; i<=rear; i++)
printf("\t%d",q[i]);}
```

**OUTPUT**






## RESULT:

Thus a C program for Queue using ADT was implemented successfully

**EX.NO:2** **ARRAY IMPLEMENTATION OF LIST ADT**

**AIM:**

.
To write a program for List using array implementation.

**DESCRIPTION:**

A linked list is a sequence of data structures, which are connected together via links. Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array.

A linked list is a sequence of data structures, which are connected together via links. Linked List is a sequence of links which contains items. Each link contains a connection to another link. Linked list is the second most-used data structure after array.

Following are the important terms to understand the concept of Linked List.

- Link − each link of a linked list can store a data called an element.
- Next − each link of a linked list contains a link to the next link called Next.
- Linked List − A Linked List contains the connection link to the first link called First.

**ALGORITHM:**

Step1: Create nodes first, last; next, prev and cur then set the value as NULL.

Step 2: Read the list operation type.

Step 3: If operation type is create then process the following steps.

1. Allocate memory for node cur.
2. Read data in cur's data area.
3. Assign cur node as NULL.
4. Assign first=last=cur.

Step 4: If operation type is Insert then process the following steps.

1. Allocate memory for node cur.
2. Read data in cur's data area.
3. Read the position the Data to be insert.
4. Availability of the position is true then assing cur's node as first and first=cur.
5. If availability of position is false then do following steps.
    1. Assign next as cur and count as zero.
    2. Repeat the following steps until count less than postion.
        1 .Assign prev as next
        2. Next as prev of node. 3. Add count by one.

4. If prev as NULL then display the message INVALID POSITION.
5. If prev not qual to NULL then do the following steps.
    1. Assign cur's node as prev's node.
    2. Assign prev's node as cur.

Step5: If operation type is delete then do the following steps.

1. Read the position .
2. Check list is Empty .If it is true display the message List empty.
3. If position is first.
    1. Assign cur as first.
    2. Assign First as first of node.
    3. Reallocate the cur from memory.
    4. If position is last.
        1. Move the current node to prev.
        2. cur's node as Null.
        3. Reallocate the Last from memory.
        4. Assign last as cur.
        5. If position is enter Mediate.
            1. Move the cur to required postion.
            2. Move the Previous to cur's previous position
            3. Move the Next to cur's Next position.
            4. Now Assign previous of node as next.
            5. Reallocate the cur from memory.

step 6: If operation is traverse.

1. Assign current as first.
2. Repeat the following steps untill cur becomes NULL.

**PROGRAM:**

```
#include<stdio.h>

#include<conio.h>

#define MAX 10

void create();

void insert();

void deletion();

void search();

void display();
```

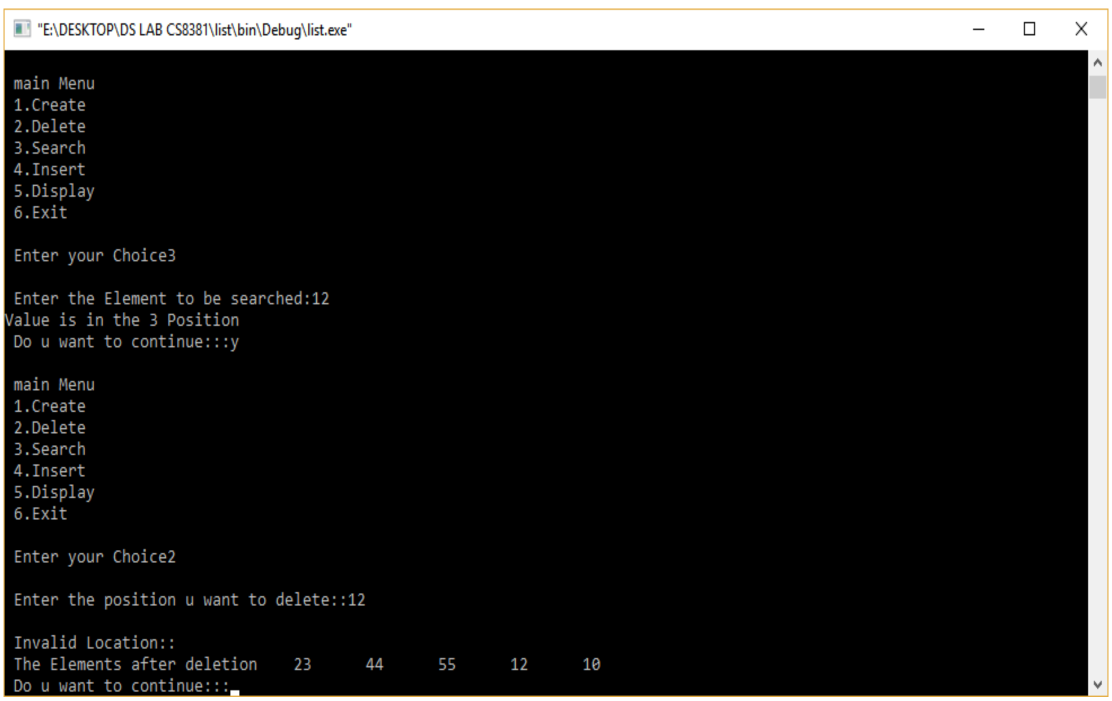
```
int a,b[20], n, p, e, f, i, pos;

void main()

{

clrscr();

int ch;

char g='y';

do

{

printf("\n main Menu");

printf("\n 1.Create \n 2.Delete \n 3.Search \n 4.Insert \n 5.Display\n 6.Exit
\n");

printf("\n Enter your Choice");

scanf("%d", &ch);

switch(ch)

{

case 1:

create();

break;

case 2:

deletion();

break;

case 3:

search();

break;

case 4:

insert();

break;

case 5:

display();

break;

case 6:

exit();

break;

default:

printf("\n Enter the correct choice:");
```

```
}
printf("\n Do u want to continue:::");
scanf("\n%c", &g);
}
while(g=='y'||g=='Y');
getch();
}
void create()
{
printf("\n Enter the number of nodes");
scanf("%d", &n);
for(i=0;i<n;i++)
{
printf("\n Enter the  Element:",i+1);
scanf("%d", &b[i]);
}
}
void deletion()
{
printf("\n Enter the position u want to delete::");
scanf("%d", &pos);
if(pos>=n)
{
printf("\n Invalid Location::");
}
else
{
for(i=pos+1;i<n;i++)
{
b[i-1]=b[i];
}
n--;
}
printf("\n The Elements after deletion");
```

```
for(i=0;i<n;i++)
{
printf("\t%d", b[i]);
}
}
void search()
{
printf("\n Enter the Element to be searched:");
scanf("%d", &e);
for(i=0;i<n;i++)
{
if(b[i]==e)
{
printf("Value is in the %d Position", i);
}}}
void insert()
{
printf("\n Enter the position u need to insert::");
scanf("%d", &pos);
if(pos>=n)
 {
 printf("\n invalid Location::");
 } else
{
 for(i=MAX-1;i>=pos-1;i--)
 {
 b[i+1]=b[i];
 }
 printf("\n Enter the element to insert::\n");
 scanf("%d",&p);
 b[pos]=p;
 n++;
 }
 printf("\n The list after insertion::\n");
```

```
display();}
void display(){
printf("\n The Elements of The list ADT are:");
for(i=0;i<n;i++)
{
printf("\n\n%d", b[i]);
}}
```

## OUTPUT


## RESULT:

Thus the C program for array implementation of List ADT was created, executed and output was verified successfully

**EX. NO: 3A** **STACK ADT USING LINKED LIST**

**AIM:**

To write a C program for stack ADT using linked list implementation.

**DESCRIPTION:**

The major problem with the stack implemented using array is, it works only for fixed number of data values. That means the amount of data must be specified at the beginning of the implementation itself. Stack implemented using array is not suitable, when we don't know the size of data which we are going to use. A stack data structure can be implemented by using linked list data structure. The stack implemented using linked list can work for unlimited number of values. That means, stack implemented using linked list works for variable size of data. So, there is no need to fix the size at the beginning of the implementation. The Stack implemented using linked list can organize as many data values as we want.

In linked list implementation of a stack, every new element is inserted as 'top' element. That means every newly inserted element is pointed by 'top'. Whenever we want to remove an element from the stack, simply remove the node which is pointed by 'top' by moving 'top' to its next node in the list. The next field of the first element must be always NULL.

**ALGORITHM:**

1. Define a struct for each node in the stack. Each node in the stack contains data and link to the next node. TOP pointer points to last node inserted in the stack.
2. The operations on the stack are
   a. PUSH data into the stack
   b. POP data out of stack
3. PUSH DATA INTO STACK
   a. Enter the data to be inserted into stack.
   b. If TOP is NULL
      i. The input data is the first node in stack.
      ii. The link of the node is NULL.
      iii. TOP points to that node.
   c. If TOP is NOT NULL
      i. The link of TOP points to the new node.
      ii. TOP points to that node.
4. POP DATA FROM STACK

   a. 4a.If TOP is NULL
      i. the stack is empty
   b. 4b.If TOP is NOT NULL
      i. The link of TOP is the current TOP.
      ii. The pervious TOP is popped from stack.
5. The stack represented by linked list is traversed to display its content.

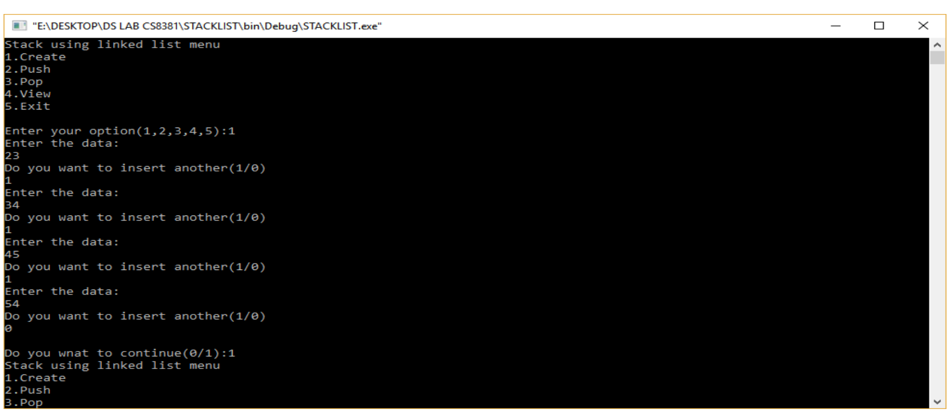
**PROGRAM:**

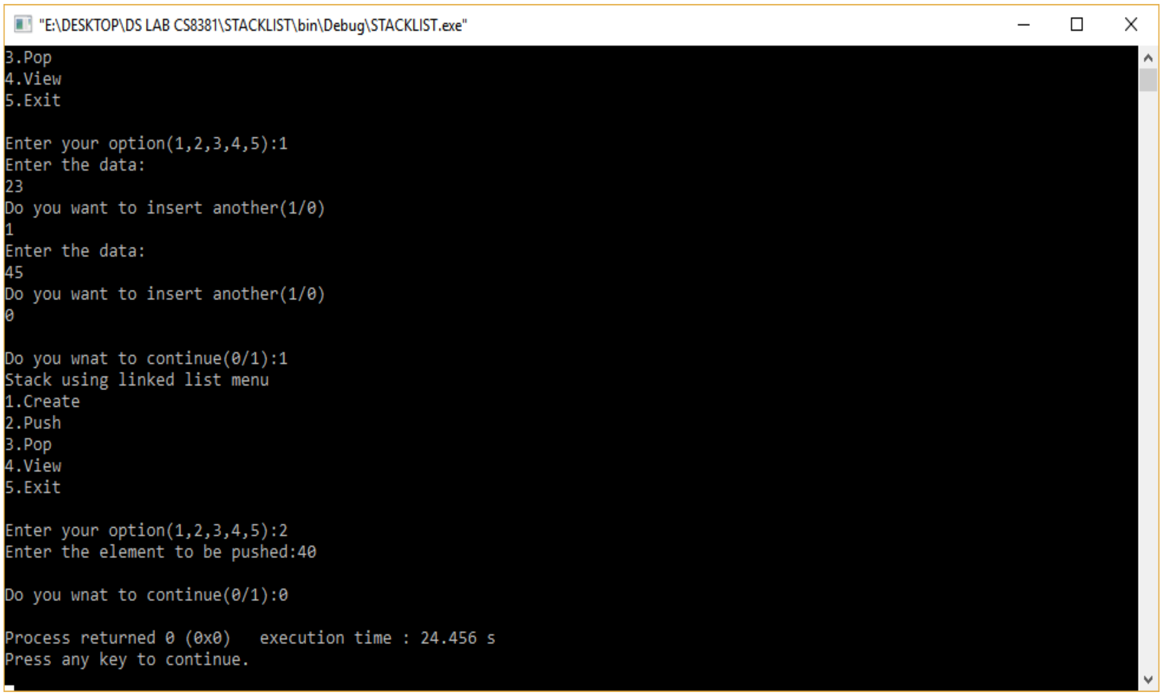
```
#include<stdio.h>
#include<conio.h>
#include<alloc.h>
struct node
{
	int data;
	struct node *next;
}*top,*new1,*first;

void main()
{
	int wish,opt;
	void create(),push(),pop(),view();
	do
	{
		clrscr();
		printf("Stack using linked list menu");
		printf("\n1.Create\n2.Push\n3.Pop\n4.View\n5.Exit\n");
		printf("\nEnter your option(1,2,3,4,5):");
		scanf("%d",&wish);
		switch(wish)
		{
			case 1: create(); break;
			case 2: push(); break;
			case 3: pop(); break;
```

```
                case 4: view(); break;
                      case 5: exit(0);
                }
                printf("\nDo you wnat to continue(0/1):");
                scanf("%d",&opt);
          }while(opt==1);
    }
    void create()
    {
          int ch;
          top=(struct node*)malloc(sizeof(struct node));
          top->next=NULL;
          do
          {
                clrscr();
                printf("Enter the data:\n");
                scanf("%d",&top->data);
                printf("Do you want to insert another(1/0)\n");
                scanf("%d",&ch);
                if(ch==1)
                {
                      new1=(struct node*)malloc(sizeof(struct node));
                      new1->next=top;
                      top=new1;
                      first=top;
                }
                else
                      break;
          }while(ch==1);
    }
    void push()
    {
          top=first;
          new1=(struct node*)malloc(sizeof(struct node));
          printf("Enter the element to be pushed:");
```

```
        scanf("%d",&new1->data);
        new1->next=top;
        top=new1;
        first=top;
}
void pop()
{
        clrscr();
        top=first;
        if(top==NULL)
        printf("\n Stack is empty");
        else
        {
                printf("\nThe element popped out from stack is %d",top->data);
                top=top->next;
                first=top;
        }}
void view()
{
        printf("\nStack contents\n");
        while(top->next!=NULL)
        {printf("%d->",top->data);
                top=top->next;}
        printf("%d\n",top->data);
        getch();}
```

## OUTPUT





```
"E:\DESKTOP\DS LAB CS8381\STACKLIST\bin\Debug\STACKLIST.exe"
3.Pop
4.View
5.Exit

Enter your option(1,2,3,4,5):1
Enter the data:
23
Do you want to insert another(1/0)
1
Enter the data:
45
Do you want to insert another(1/0)
0

Do you wnat to continue(0/1):1
Stack using linked list menu
1.Create
2.Push
3.Pop
4.View
5.Exit

Enter your option(1,2,3,4,5):2
Enter the element to be pushed:40

Do you wnat to continue(0/1):0

Process returned 0 (0x0)   execution time : 24.456 s
Press any key to continue.
```

## RESULT:

Thus the C program for array implementation of Stack ADT was created, executed and output was verified successfully

**EX. NO :3B** **QUEUE ADT USING LINKED LIST**

**AIM:**

To write a C program for Queue using Linked implementation.

**DESCRIPTION:**

Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data(enqueue) and the other is used to remove data(dequeue). Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed first.

Queue is an abstract data structure, somewhat similar to Stacks. Unlike stacks, a queue is open at both its ends. One end is always used to insert data (enqueue) and the other is used to remove data (dequeue). Queue follows First-In-First-Out methodology, i.e., the data item stored first will be accessed first.

Queue operations may involve initializing or defining the queue, utilizing it, and then completely erasing it from the memory. Here we shall try to understand the basic operations associated with queues.

- enqueue () − add (store) an item to the queue.
- dequeue () − remove (access) an item from the queue.

Few more functions are required to make the above-mentioned queue operation efficient. These are

- peek() − Gets the element at the front of the queue without removing it.
- isfull() − Checks if the queue is full.
- isempty() − Checks if the queue is empty.

**ALGORITHM:**

1. Define a struct for each node in the queue. Each node in the queue contains data and link to the next node. Front and rear pointer points to first and last node inserted in the queue.
2. The operations on the queue are
   a. INSERT data into the queue
   b. DELETE data out of queue
3. INSERT DATA INTO queue
   a. Enter the data to be inserted into queue.

b. If TOP is NULL

i. The input data is the first node in queue. ii.

The link of the node is NULL.

iii. TOP points to that node. c. If

TOP is NOT NULL

i. The link of TOP points to the new node. ii.

TOP points to that node.

4. DELETE DATA FROM queue a. If

TOP is NULL

i. the queue is empty b. If

TOP is NOT NULL

i. The link of TOP is the current TOP.

ii. The pervious TOP is popped from queue.

5. The queue represented by linked list is traversed to display its content.

**PROGRAM:**

```
#include<stdio.h>

#include<conio.h>

struct node

{

int info;

struct node *link;

}*front = NULL, *rear = NULL;


void insert();

void delet();

void display();

int item;

void main()

{


int ch;

do

{

printf("\n\n1.\tEnqueue\n2.\tDequeue\n3.\tDisplay\n4.\tExit\n");
```

```
printf("\nEnter your choice: ");
scanf("%d", &ch);
switch(ch)
{
case 1:
insert();
break;
case 2:
delet();
break;
case 3:
display();
break;
case 4:
exit(0);
default:
printf("\n\nInvalid choice. Please try again...\n");
}
} while(1);
getch();
}

void insert()
{
printf("\n\nEnter ITEM: ");
scanf("%d", &item);
if(rear == NULL)
{
rear = (struct node *)malloc(sizeof(struct node));
rear->info = item;
rear->link = NULL;
front = rear;
}
else{
```

```
rear->link = (struct node *)malloc(sizeof(struct node));
rear = rear->link;
rear->info = item;
rear->link = NULL;
}}
void delet(){
struct node *ptr;
if(front == NULL)
printf("\n\nQueue is empty.\n");
else{
ptr = front;
item = front->info;
front = front->link;
free(ptr);
printf("\nItem deleted: %d\n", item);
if(front == NULL)
rear = NULL;
}}
void display()
{
struct node *ptr = front;
if(rear == NULL)
printf("\n\nQueue is empty.\n");
else
{
printf("\n\n");
while(ptr != NULL)
{
printf("%d\t",ptr->info);
ptr = ptr->link;
}}}
```

## OUTPUT

"E:\DESKTOP\DS LAB CS8381\QUEUEADT\bin\Debug\QUEUEADT.exe"

```
1.      Enqueue
2.      Dequeue
3.      Display
4.      Exit

Enter your choice: 1


Enter ITEM: 23


1.      Enqueue
2.      Dequeue
3.      Display
4.      Exit

Enter your choice: 1


Enter ITEM: 34


1.      Enqueue
2.      Dequeue
3.      Display
4.      Exit

Enter your choice: 3
```

"E:\DESKTOP\DS LAB CS8381\QUEUEADT\bin\Debug\QUEUEADT.exe"

```
1.      Enqueue
2.      Dequeue
3.      Display
4.      Exit

Enter your choice: 2

Item deleted: 23


1.      Enqueue
2.      Dequeue
3.      Display
4.      Exit

Enter your choice: 3


34

1.      Enqueue
2.      Dequeue
3.      Display
4.      Exit

Enter your choice: 4

Process returned 0 (0x0)   execution time : 68.865 s
Press any key to continue.
```

## RESULT:

Thus the C program for array implementation of Queue ADT was created, executed and output was verified successfully

## EXNO:4A REPRESENT A POLYNOMIAL AS A LINKED LIST

**AIM:**

To write program in C to convert given infix expression in to postfix notation

**DESCRIPTION:**

A polynomial is homogeneous ordered list of pairs <exponent, coefficient>, where each coefficient is unique.

Example:

$3x^2+5x+7$

Linked list representation

The main fields of polynomial are coefficient and exponent, in linked list it will have one more filed called „link" field to point to next term in the polynomial. If there are „n" terms in the polynomial then „n" such nodes have to be created.

**ALGORITHM:**

1: Get the two polynomials. First polynomial is P1 and second polynomial is P2

2: For addition of two polynomials if exponents of both the polynomials are same then we ad the coefficients. For storing the result we will create the third linked lists say P3.

3: If Exponent of P2 is greater than exponent of P1 then keep the P3 as P2.

4: If Exponent of P2 is greater than exponent of P1 then keep the P3 as P1

5: If Exponent of P2 is equal to the exponent of P1 then add the coefficient of P1 and coefficient of P2 as coefficient of P3.

6: Continue the above step from 3 to 5 until end o the two polynomials.

7: If any of the polynomial is ended keep P3 as the remaining polynomial.

8: Stop the execution.

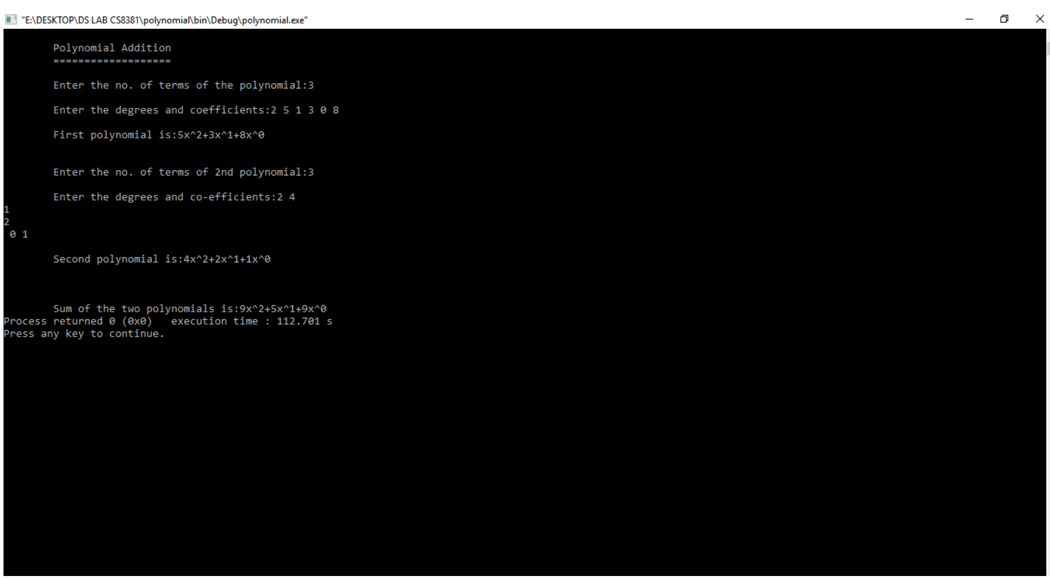
**PROGRAM:**

```
#include<stdio.h>
#include<conio.h>

main()
{
int a[10], b[10], c[10],m,n,k,k1,i,j,x;

clrscr();

printf("\n\tPolynomial Addition\n");
printf("\t===================\n");
printf("\n\tEnter the no. of terms of the polynomial:");
scanf("%d", &m);
printf("\n\tEnter the degrees and coefficients:");
for (i=0;i<2*m;i++)
scanf("%d", &a[i]);
```

```
printf("\n\tFirst polynomial is:");
 k1=0;
if(a[k1+1]==1)
printf("x^%d", a[k1]);
else
printf("%dx^%d", a[k1+1],a[k1]);
k1+=2;

while (k1<i)
{
printf("+%dx^%d", a[k1+1],a[k1]);
k1+=2;
}
printf("\n\n\n\tEnter the no. of terms of 2nd polynomial:");
scanf("%d", &n);
 printf("\n\tEnter the degrees and co-efficients:");
for(j=0;j<2*n;j++)
scanf("%d", &b[j]);
printf("\n\tSecond polynomial is:");
k1=0;
if(b[k1+1]==1)
printf("x^%d", b[k1]);
else
printf("%dx^%d",b[k1+1],b[k1]);
k1+=2;
while (k1<2*n)
{
printf("+%dx^%d", b[k1+1],b[k1]);
k1+=2;
}
i=0;
j=0;
k=0;

while (m>0 && n>0)
{
if (a[i]==b[j])
{
c[k+1]=a[i+1]+b[j+1];
c[k]=a[i];
m--;
n--;
i+=2;
j+=2;
}
else if (a[i]>b[j])
{
c[k+1]=a[i+1];
c[k]=a[i];
m--;
i+=2;
}
else
{
c[k+1]=b[j+1];
c[k]=b[j];
n--;
j+=2;
}
k+=2;
}
while (m>0)
{
c[k+1]=a[i+1];
c[k]=a[i];
```

```
k+=2;
i+=2;
m--;
}

while (n>0)
{
c[k+1]=b[j+1];
c[k]=b[j];
k+=2;
j+=2;
n--;
}
printf("\n\n\n\n\tSum of the two polynomials is:");
k1=0;
if (c[k1+1]==1)
printf("x^%d", c[k1]);
else
printf("%dx^%d", c[k1+1],c[k1]);
k1+=2;
while (k1<k)
{
if (c[k1+1]==1)
printf("+x^%d", c[k1]);
else
printf("+%dx^%d", c[k1+1], c[k1]);
k1+=2;
}
getch();
return 0;

}
```

**OUTPUT**




**RESULT:**

Thus the program in C to convert given infix expression in to postfix notation

## EX.NO:4B CONVERSION OF INFIX EXPRESSION TO POSTFIX NOTATION

### AIM:

To write program in C to convert given infix expression to postfix notation

### ALGORITHM:

1: Get an infix expression.

2: Scan the expression from left to right.

3: If any operands come display it.

4: If the incoming symbol in a operator and has more priority then the symbol into the stack.

5: If the incoming operator has less priority than the stack symbol then copy the symbol at the top of the stack and then print until the condition becomes false and push the following operator on the stack.

6: If the symbol is ')' then copy operators from top of the stack. Deletion opening parenthesis is from top of the stack.

7: Stop the process.

### DESCRIPTION:

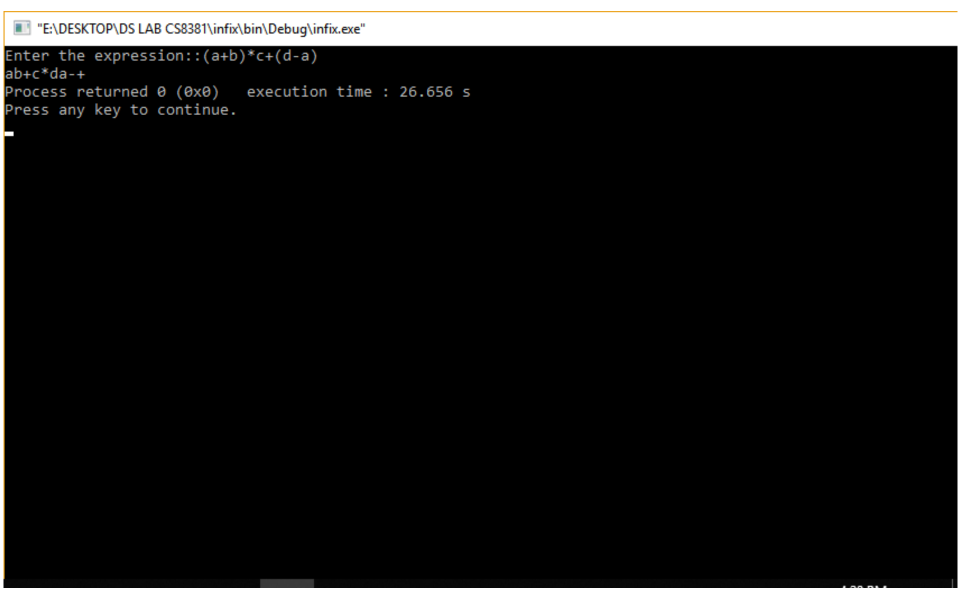
**Infix expression:**The expression of the form a op b. When an operator is in-between every pair of operands.
**Postfix expression:**The expression of the form a b op. When an operator is followed for every pair of operands.

### PROGRAM:

```
#include<stdio.h>
char stack[20];
int top = -1;
void push(char x)
{
    stack[++top] = x;
}

char pop()
{
    if(top == -1)
        return -1;
    else
        return stack[top--];
}
int priority(char x)
{
    if(x == '(')
        return 0;
    if(x == '+' || x == '-')
        return 1;
    if(x == '*' || x == '/')
```

```
return 2;
}
main()
{
    char exp[20];
    char *e, x;
    printf("Enter the expression :: ");
    scanf("%s",exp);
    e = exp;
    while(*e != '\0')
    {
        if(isalnum(*e))
            printf("%c",*e);
        else if(*e == '(')
            push(*e);
        else if(*e == ')')
        {
            while((x = pop()) != '(')
                printf("%c", x);
        }
        else
        {
            while(priority(stack[top]) >= priority(*e))
                printf("%c",pop());
            push(*e);
        }
        e++;
    }
    while(top != -1)
    {
        printf("%c",pop());
    }
}
```

**OUTPUT**




**RESULT:**

Thus the program in C to convert given infix expression to postfix notation

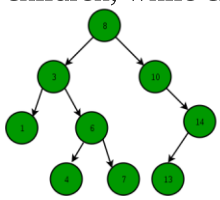
**EXNO:5 IMPLEMENTATION BINARY TREE AND OPERATIONS OF BINARY TREES**

**AIM:**

To write a C program Implementation Binary Tree And Operations Of Binary Trees

## DESCRIPTION:

A binary tree is a tree data structure where each node has up to two child nodes, creating the branches of the tree. The two children are usually called the left and right nodes. Parent nodes are nodes with children, while child nodes may include references to their parents.



## ALGORITHM

1.Start from root.

2. Compare the inserting element with root, if less than root, then recurse for left, else recurse for right.

3. If element to search is found anywhere, return true, else return false

## PROGRAM

```
#include<stdio.h>
#include<stdlib.h>

struct tree {
      int data;
      struct tree *left;
      struct tree *right;
} *root = NULL, *node = NULL, *temp = NULL;

struct tree* insert(int key,struct tree *leaf) {
      if(leaf == 0) {
            struct tree *temp;
            temp = (struct tree *)malloc(sizeof(struct tree));
            temp->data = key;
            temp->left = 0;
            temp->right = 0;
            printf("Data inserted!\n");
            return temp;
      }
      else {
            if(key < leaf->data)
                  leaf->left = insert(key,leaf->left);
            else
                  leaf->right = insert(key,leaf->right);
      }
      return leaf;
}


struct tree* search(int key,struct tree *leaf) {
      if(leaf != NULL) {
            if(key == leaf->data) {
                  printf("Data found!\n");
                  return leaf;
            }
            else {
                  if(key < leaf->data)
                        return search(key,leaf->left);
                  else
```

```
                        return search(key,leaf->right);
                }
        }


else {
                printf("Data not found!\n");return NULL;


}}

struct tree* minvalue(struct tree *node) {
        if(node == NULL)
                return NULL;

        if(node->left)
                return minvalue(node->left);
        else
                return node;
}

/* Function for find maximum value from the Tree */
struct tree* maxvalue(struct tree *node) {
        if(node == NULL)
                return NULL;


        if(node->right)
                return maxvalue(node->right);
        else
                return node;
}

void preorder(struct tree *leaf) {
        if(leaf == NULL)
                return;
        printf("%d\n",leaf->data);
        preorder(leaf->left);
        preorder(leaf->right);
}


void inorder(struct tree *leaf) {
        if(leaf == NULL)
                return;
        preorder(leaf->left);
        printf("%d\n",leaf->data);
        preorder(leaf->right);
}


void postorder(struct tree *leaf) {
        if(leaf == NULL)
                return;
        preorder(leaf->left);
        preorder(leaf->right);
        printf("%d\n",leaf->data);
}


struct tree* delete(struct tree *leaf, int key) {
        if(leaf == NULL)
                printf("Element Not Found!\n");
        else if(key < leaf->data)
                leaf->left = delete(leaf->left, key);
        else if(key > leaf->data)
                leaf->right = delete(leaf->right, key);
        else {
                if(leaf->right && leaf->left) {


                        temp = minvalue(leaf->right);
                        leaf->data = temp->data;
```

```
                    leaf->right = delete(leaf->right,temp->data);


        }
                else {

                    temp = leaf;
                    if(leaf->left == NULL)
                        leaf = leaf->right;
                    else if(leaf->right == NULL)
                        leaf = leaf->left;
                    free(temp);
                    printf("Data delete successfully!\n");
                }
            }
        }

        int main() {
            int key, choice;
            while(choice != 7) {
                printf("1. Insert\n2. Search\n3. Delete\n4. Display\n5. Min Value\n6.
        Max Value\n7. Exit\n");
                printf("Enter your choice:\n");
                scanf("%d", &choice);
                switch(choice) {
                    case 1:
                        printf("\nEnter the value to insert:\n");
                        scanf("%d", &key);
                        root = insert(key, root);
                        break;
                    case 2:
                        printf("\nEnter the value to search:\n");
                        scanf("%d", &key);
                        search(key,root);
                        break;
                    case 3:
                        printf("\nEnter the value to delete:\n");
                        scanf("%d", &key);
                        delete(root,key);
                        break;
                    case 4:
                        printf("Preorder:\n");
                        preorder(root);
                        printf("Inorder:\n");
                        inorder(root);
                        printf("Postorder:\n");
                        postorder(root);
                        break;
                    case 5:
                        if(minvalue(root) == NULL)
                            printf("Tree is empty!\n");
                        else
                            printf("Minimum value is %d\n", minvalue(root)-
        >data);
                        break;
                    case 6:
                        if(maxvalue(root) == NULL)
                            printf("Tree is empty!\n");
                        else
                            printf("Maximum value is %d\n", maxvalue(root)-
        >data);
                        break;
                    case 7:
                        printf("Bye Bye!\n");
                        exit(0);
                        break;
                    default:
                        printf("Invalid choice!\n");
                }
            }
            return 0;
```

}

## OUTPUT

```
"E:\DESKTOP\DS LAB CS8381\binarytree search\bin\Debug\binarytree search.exe"
1. Insert
2. Search
3. Delete
4. Display
5. Min Value
6. Max Value
7. Exit
Enter your choice:
1

Enter the value to insert:
2
Data inserted!
1. Insert
2. Search
3. Delete
4. Display
5. Min Value
6. Max Value
7. Exit
Enter your choice:
1

Enter the value to insert:
3
Data inserted!
1. Insert
2. Search
3. Delete
4. Display
```

```
"E:\DESKTOP\DS LAB CS8381\binarytree search\bin\Debug\binarytree search.exe"
Enter the value to insert:
5
Data inserted!
1. Insert
2. Search
3. Delete
4. Display
5. Min Value
6. Max Value
7. Exit
Enter your choice:
4
Preorder:
2
3
5
Inorder:
2
3
5
Postorder:
3
5
2
1. Insert
2. Search
3. Delete
4. Display
5. Min Value
6. Max Value
```

## Result:

Thus the program in C is implementated Binary Tree and Operations of Binary Trees.

## EX. NO: 6 IMPLEMENTATION OF BINARY SEARCH TREE

**AIM:**

To write a C program to implementation of binary search tree.

**DESCRIPTION:**

A Binary Search Tree (BST) is a tree in which all the nodes follow the below-mentioned properties

The left sub-tree of a node has a key less than or equal to its parent node's key.

The right sub-tree of a node has a key greater than to its parent node's key.

Thus, BST divides all its sub-trees into two segments; the left sub-tree and the right sub-tree and can be defined as

left_subtree (keys) $\leq$ node (key) $\leq$ right_subtree (keys)

Representation

BST is a collection of nodes arranged in a way where they maintain BST properties. Each node has a key and an associated value. While searching, the desired key is compared to the keys in BST and if found, the associated value is retrieved.

Following is a pictorial representation of BST

Basic Operations

Following are the basic operations of a tree

- Search − Searches an element in a tree.
- Insert − Inserts an element in a tree.
- Pre-order Traversal − Traverses a tree in a pre-order manner.
- In-order Traversal − Traverses a tree in an in-order manner.
- Post-order Traversal − Traverses a tree in a post-order manner.

**ALGORITHM:**

1. Declare function create (), search (), delete (), Display ().
2. Create a structure for a tree contains left pointer and right pointer.
3. Insert an element is by checking the top node and the leaf node and the operation will be performed.
4. Deleting an element contains searching the tree and deleting the item.
5. Display the Tree elements.

**PROGRAM:**

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
#include<alloc.h>

struct tree
{
	int data;
	struct tree *lchild;
	struct tree *rchild;
}*t,*temp;

int element;
void inorder(struct tree *);
void preorder(struct tree *);
void postorder(struct tree *);
struct tree * create(struct tree *, int);
struct tree * find(struct tree *, int);
struct tree * insert(struct tree *, int);
struct tree * del(struct tree *, int);
struct tree * findmin(struct tree *);
struct tree * findmax(struct tree *);
void main()
{
	int ch;

	do
	{
		printf("\n\t\t\tBINARY SEARCH TREE");
		printf("\n\t\t\t****** ****** ****");
		printf("\nMain Menu\n");
		printf("\n1.Create\n2.Insert\n3.Delete\n4.Find\n5.FindMin\n6.FindMax")
	;
		printf("\n7.Inorder\n8.Preorder\n9.Postorder\n10.Exit\n");
		printf("\nEnter ur choice :");
		scanf("%d",&ch);
		switch(ch)
		{
			case 1:
				printf("\nEnter the data:");
				scanf("%d",&element);
				t=create(t,element);
				inorder(t);
```

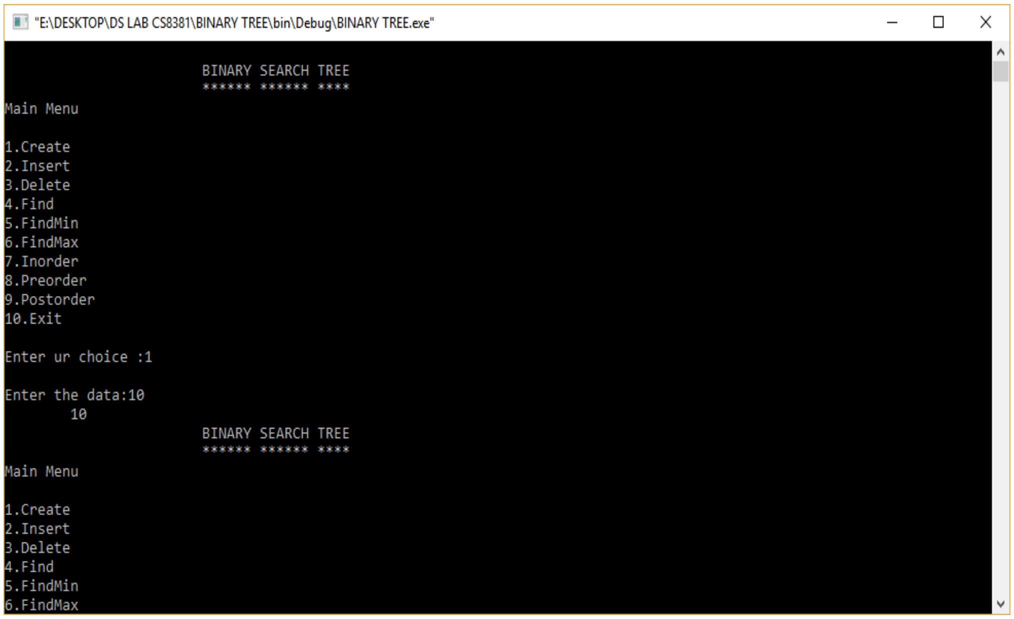
```
                    break;
                case 2:
}                   printf("\nEnter the data:");
                    scanf("%d",&element);
                    t=insert(t,element);
                    inorder(t);
                    break;
                case 3:
                    printf("\nEnter the data:");
                    scanf("%d",&element);
                    t=del(t,element);
                    inorder(t);
                    break;
                case 4:
                    printf("\nEnter the data:");
                    scanf("%d",&element);
                    temp=find(t,element);
                    if(temp->data==element)
                        printf("\nElement %d is at %d",element,temp);
                    else
                        printf("\nElement is not found");
                    break;
                case 5:
                    temp=findmin(t);
                    printf("\nMax element=%d",temp->data);
                    break;
                case 6:
                    temp=findmax(t);
                    printf("\nMax element=%d",temp->data);
                    break;
                case 7:
                    inorder(t);
                    break;
                case 8:
                    preorder(t);
                    break;
                case 9:
                    postorder(t);
                    break;
                case 10:
                    exit(0);
            }
        }while(ch<=10);
}
```

```
struct tree * create(struct tree *t, int element)
{
      t=(struct tree *)malloc(sizeof(struct tree));
      t->data=element;
      t->lchild=NULL;
      t->rchild=NULL;
      return t;
}
struct tree * find(struct tree *t, int element)
{
      if(t==NULL)
            return NULL;
      if(element<t->data)
            return(find(t->lchild,element));
      else
            if(element>t->data)
                  return(find(t->rchild,element));
            else
                  return t;
}
struct tree *findmin(struct tree *t)
{
      if(t==NULL)
            return NULL;
      else
            if(t->lchild==NULL)
                  return t;
            else
                  return(findmin(t->lchild));
}
struct tree *findmax(struct tree *t)
{
      if(t!=NULL)


      {
            while(t->rchild!=NULL)
                  t=t->rchild;
      }
      return t;
}
struct tree *insert(struct tree *t,int element)
{
      if(t==NULL)
      {
```

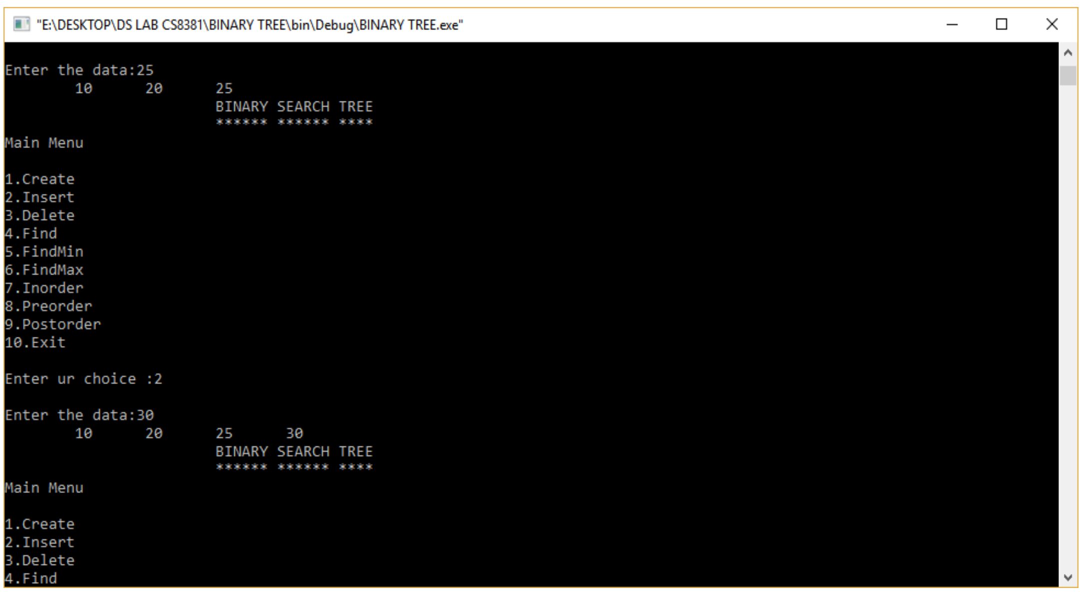
```
            t=(struct tree *)malloc(sizeof(struct tree));
            t->data=element;
            t->lchild=NULL;
            t->rchild=NULL;
            return t;
        }
        else
        {
            if(element<t->data)
            {
                t->lchild=insert(t->lchild,element);
            }
            else
                if(element>t->data)
                {
                    t->rchild=insert(t->rchild,element);
                }
                else
                if(element==t->data)
                {
                    printf("element already present\n");
                }
                return t;
        }
}
struct tree * del(struct tree *t, int element)
{
        if(t==NULL)
            printf("element not found\n");

        else
            if(element<t->data)
                t->lchild=del(t->lchild,element);
            else
                if(element>t->data)
                    t->rchild=del(t->rchild,element);
                else
                    if(t->lchild&&t->rchild)
                    {
                        temp=findmin(t->rchild);
                        t->data=temp->data;
                        t->rchild=del(t->rchild,t->data);
                    }
                    else
```

```
                        {
                              temp=t;
                              if(t->lchild==NULL)
                                    t=t->rchild;
                              else
                                    if(t->rchild==NULL)
                                          t=t->lchild;
                                    free(temp);
                        }
      return t;
}

void inorder(struct tree *t)
{
      if(t==NULL)
            return;
      else
      {
            inorder(t->lchild);
            printf("\t%d",t->data);
            inorder(t->rchild);
      }
}
void preorder(struct tree *t)
{
      if(t==NULL)
            return;
      else
      {
            printf("\t%d",t->data);
            preorder(t->lchild);
            preorder(t->rchild);
      }
}

void postorder(struct tree *t)
{
      if(t==NULL)
            return;
      else
      {
            postorder(t->lchild);
            postorder(t->rchild);
            printf("\t%d",t->data);}}
```

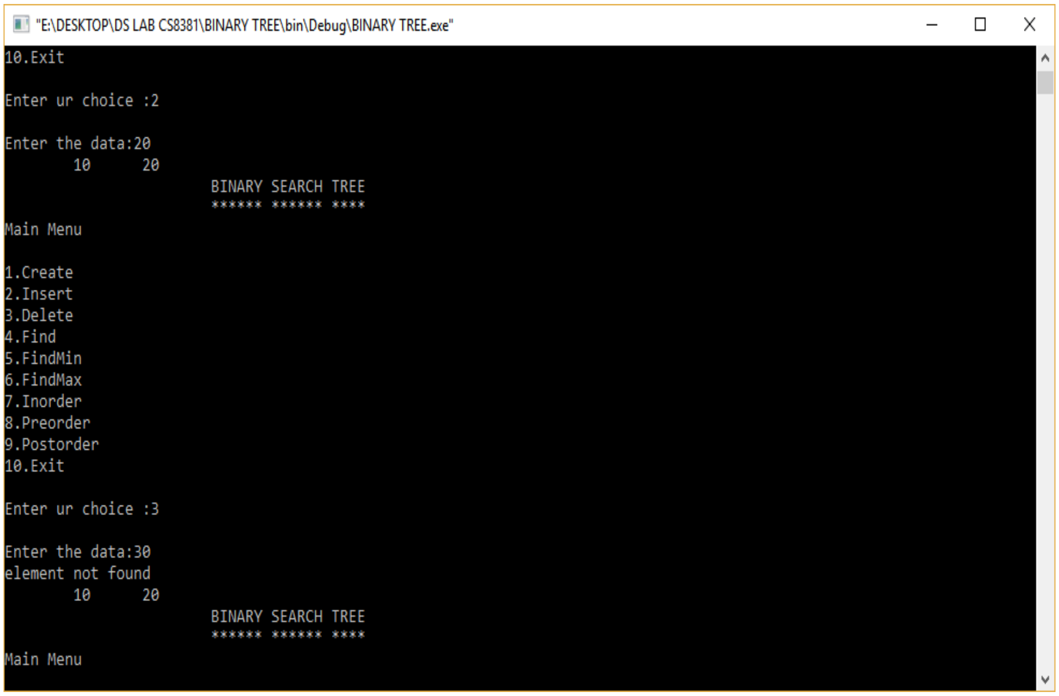
**OUTPUT:**



```
"E:\DESKTOP\DS LAB CS8381\BINARY TREE\bin\Debug\BINARY TREE.exe"

                        BINARY SEARCH TREE
                        ****** ****** ****
Main Menu

1.Create
2.Insert
3.Delete
4.Find
5.FindMin
6.FindMax
7.Inorder
8.Preorder
9.Postorder
10.Exit

Enter ur choice :1

Enter the data:10
        10
                        BINARY SEARCH TREE
                        ****** ****** ****
Main Menu

1.Create
2.Insert
3.Delete
4.Find
5.FindMin
6.FindMax
```



```
"E:\DESKTOP\DS LAB CS8381\BINARY TREE\bin\Debug\BINARY TREE.exe"

Enter the data:25
        10      20      25
                        BINARY SEARCH TREE
                        ****** ****** ****
Main Menu

1.Create
2.Insert
3.Delete
4.Find
5.FindMin
6.FindMax
7.Inorder
8.Preorder
9.Postorder
10.Exit

Enter ur choice :2

Enter the data:30
        10      20      25      30
                        BINARY SEARCH TREE
                        ****** ****** ****
Main Menu

1.Create
2.Insert
3.Delete
4.Find
```




## RESULT:

Thus the C program for binary search tree was created, executed and output was verified successfully.

**EX NO:7** **IMPLEMENTATION OF AVL TREE**

**AIM:-**

To write a C program to implement insertion in AVL trees.

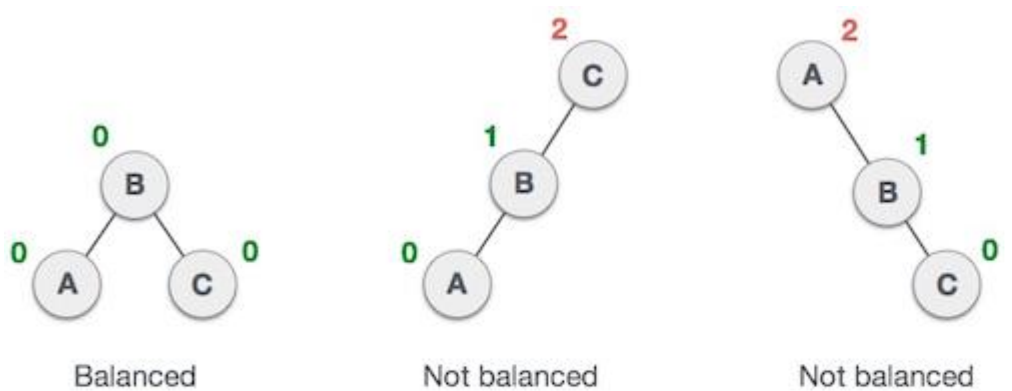
**ALGORITHM:-**

1.Initialize all variables and functions.
2.To insert and element read the value.
3.Check whether root is null
4.If yes make the new value as root.
5.Else check whether the value is equal to root value.
6.if yes, print "duplicate value".
7.Otherwise insert the value at its proper position and balance the tree using rotations.
8.To display the tree values check whether the tree is null.
9.If yes, print "tree is empty".
10.Else print all the values in the tree form and in order ofthe tree.
11.Repeat the steps 2 to 10 for more values.
12.End

**DESCRIPTION:**

**Adelson**, **Velski** & **Landis**, **AVL trees** are height balancing binary search tree. AVL tree checks the height of the left and the right sub-trees and assures that the difference is not more than 1. This difference is called the **Balance Factor**.

Here we see that the first tree is balanced and the next two trees are not balanced −




In the second tree, the left subtree of **C** has height 2 and the right subtree has height 0, so the difference is 2. In the third tree, the right subtree of **A** has height 2 and the left is missing, so it is 0, and the difference is 2 again. AVL tree permits difference (balance factor) to be only 1.

***BalanceFactor*** = height(left-sutree) − height(right-sutree)

If the difference in the height of left and right sub-trees is more than 1, the tree is balanced using some rotation techniques.

*AVL Rotations*

To balance itself, an AVL tree may perform the following four kinds of rotations −

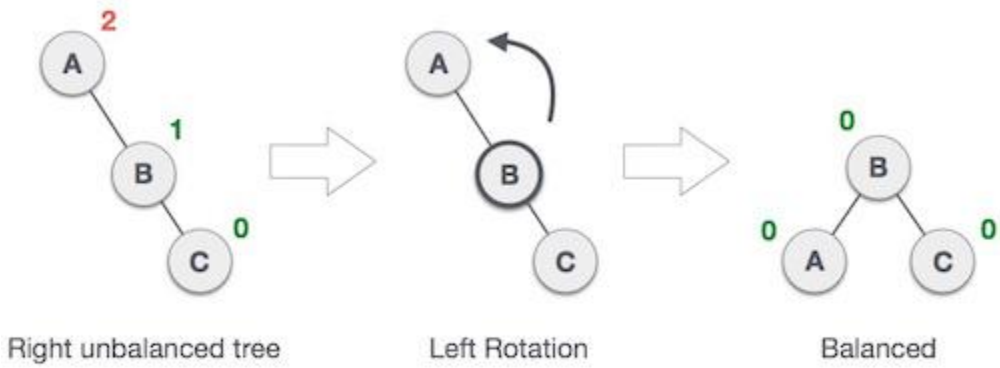
- Left rotation

- Right rotation
- Left-Right rotation
- Right-Left rotation

The first two rotations are single rotations and the next two rotations are double rotations. To have an unbalanced tree, we at least need a tree of height 2. With this simple tree, let's understand them one by one.

Left Rotation

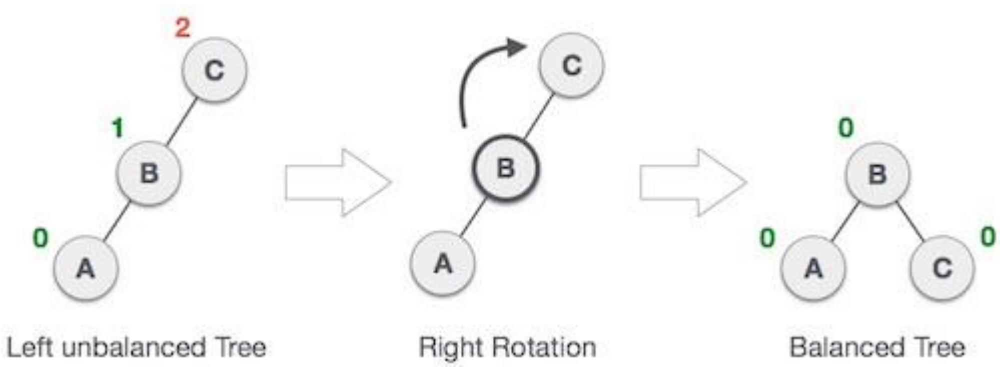
If a tree becomes unbalanced, when a node is inserted into the right subtree of the right subtree, then we perform a single left rotation −




In our example, node **A** has become unbalanced as a node is inserted in the right subtree of A's right subtree. We perform the left rotation by making **A**the left-subtree of B.

*Right Rotation*

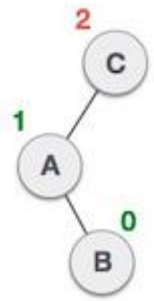
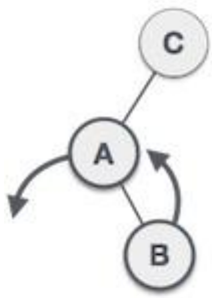
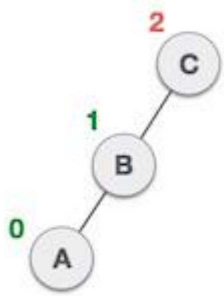
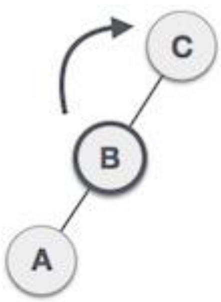
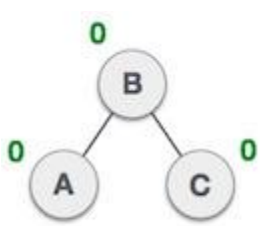
AVL tree may become unbalanced, if a node is inserted in the left subtree of the left subtree. The tree then needs a right rotation.




As depicted, the unbalanced node becomes the right child of its left child by performing a right rotation.

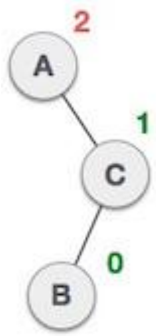
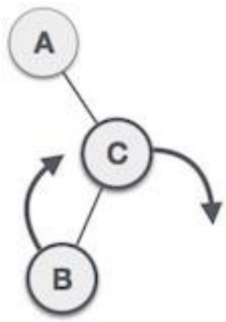
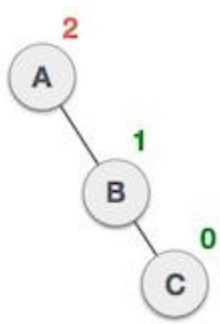
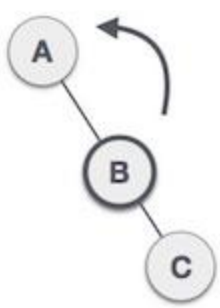
Left-Right Rotation

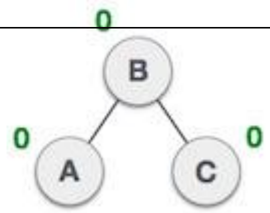
Double rotations are slightly complex version of already explained versions of rotations. To understand them better, we should take note of each action performed while rotation. Let's first check how to perform Left-Right rotation. A left-right rotation is a combination of left rotation followed by right rotation.

| State | Action |
|---|---|
|  | A node has been inserted into the right subtree of the left subtree. This makes **C** an unbalanced node. These scenarios cause AVL tree to perform left-right rotation. |
|  | We first perform the left rotation on the left subtree of **C**. This makes **A**, the left subtree of **B**. |
|  | Node **C** is still unbalanced, however now, it is because of the left-subtree of the left-subtree. |
|  | We shall now right-rotate the tree, making **B** the new root node of this subtree. **C** now becomes the right subtree of its own left subtree. |
|  | The tree is now balanced. |

Right-Left Rotation

The second type of double rotation is Right-Left Rotation. It is a combination of right rotation followed by left rotation.

| State | Action |
|---|---|
|  | A node has been inserted into the left subtree of the right subtree. This makes **A**, an unbalanced node with balance factor 2. |
|  | First, we perform the right rotation along **C** node, making **C** the right subtree of its own left subtree **B**. Now, **B** becomes the right subtree of **A**. |
|  | Node **A** is still unbalanced because of the right subtree of its right subtree and requires a left rotation. |
|  | A left rotation is performed by making **B** the new root node of the subtree. **A** becomes the left subtree of its right subtree **B**. |

| | |
|---|---|
|  | The tree is now balanced. |

**PROGRAM**

```
#include<stdio.h>

#include<malloc.h>

typedef enum { FALSE ,TRUE } bool;

struct node

{

int info;

int balance;

struct node *lchild;

struct node *rchild;

};

struct node *insert (int , struct node *, int *);

struct node* search(struct node *,int);

main()

{

bool ht_inc;

int info ;

int choice;

struct node *root = (struct node *)malloc(sizeof(struct node));

root = NULL;

while(1)

{

printf("1.Insert\n");

printf("2.Display\n");

printf("3.Quit\n");

printf("Enter your choice : ");
```

```
scanf("%d",&choice);
switch(choice)
{
case 1:
printf("Enter the value to be inserted : ");
scanf("%d", &info);
if( search(root,info) == NULL )
root = insert(info, root, &ht_inc);
else
printf("Duplicate value ignored\n");
break;
case 2:
if(root==NULL)
{
printf("Tree is empty\n");
continue;
}
printf("Tree is :\n");
display(root, 1);
printf("\n\n");
printf("Inorder Traversal is: ");
inorder(root);
printf("\n");
break;
case 3:
exit(1);
default:
printf("Wrong choice\n");
}/*End of switch*/
}/*End of while*/
```

```
}/*End of main()*/

struct node* search(struct node *ptr,int info)

{

if(ptr!=NULL)

if(info < ptr->info)

ptr=search(ptr->lchild,info);

else if( info > ptr->info)

ptr=search(ptr->rchild,info);

return(ptr);

}/*End of search()*/

struct node *insert (int info, struct node *pptr, int *ht_inc)

{

struct node *aptr;

struct node *bptr;

if(pptr==NULL)

{

pptr = (struct node *) malloc(sizeof(struct node));

pptr->info = info;

pptr->lchild = NULL;

pptr->rchild = NULL;

pptr->balance = 0;

*ht_inc = TRUE;

return (pptr);

}

if(info < pptr->info)

{

pptr->lchild = insert(info, pptr->lchild, ht_inc);

if(*ht_inc==TRUE)

{

switch(pptr->balance)

{
```

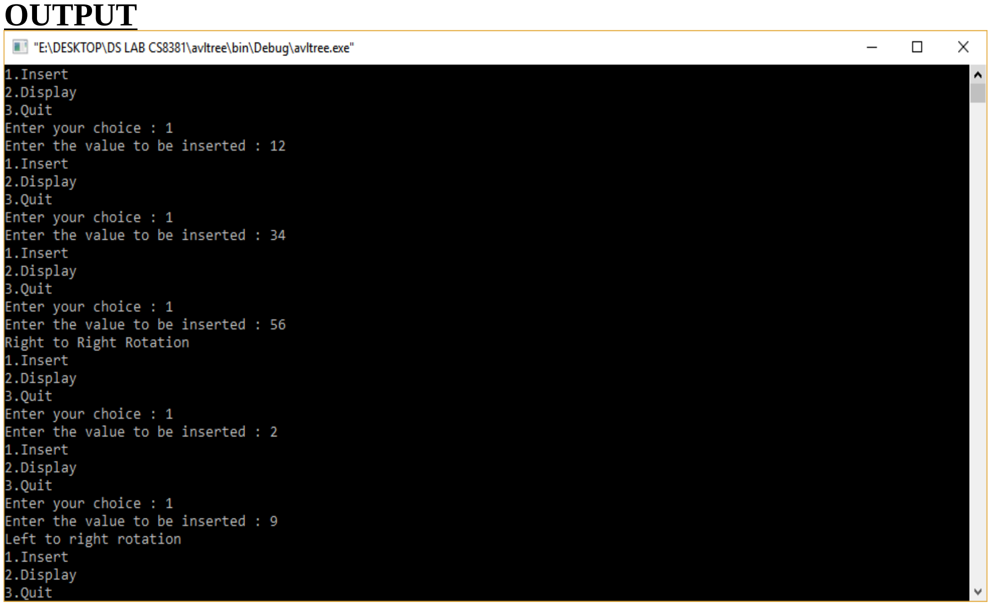
```
case -1: /* Right heavy */
pptr->balance = 0;
*ht_inc = FALSE;
break;
case 0: /* Balanced */
pptr->balance = 1;
break;
case 1: /* Left heavy */
aptr = pptr->lchild;
if(aptr->balance == 1)
{
printf("Left to Left Rotation\n");
pptr->lchild= aptr->rchild;
aptr->rchild = pptr;
pptr->balance = 0;
aptr->balance=0;
pptr = aptr;
}
else
{
printf("Left to right rotation\n");
bptr = aptr->rchild;
aptr->rchild = bptr->lchild;
bptr->lchild = aptr;
pptr->lchild = bptr->rchild;
bptr->rchild = pptr;
if(bptr->balance == 1 )
pptr->balance = -1;
else
pptr->balance = 0;
if(bptr->balance == -1)
```

```
aptr->balance = 1;
else
aptr->balance = 0;
bptr->balance=0;
pptr=bptr;
}
*ht_inc = FALSE;
}/*End of switch */
}/*End of if */
}/*End of if*/
if(info > pptr->info)
{
pptr->rchild = insert(info, pptr->rchild, ht_inc);
if(*ht_inc==TRUE)
{
switch(pptr->balance)
{
case 1: /* Left heavy */
pptr->balance = 0;
*ht_inc = FALSE;
break;
case 0: /* Balanced */
pptr->balance = -1;
break;
case -1: /* Right heavy */
aptr = pptr->rchild;
if(aptr->balance == -1)
{
printf("Right to Right Rotation\n");
pptr->rchild= aptr->lchild;
aptr->lchild = pptr;
```

```
pptr->balance = 0;
aptr->balance=0;
pptr = aptr;
}
else
{
printf("Right to Left Rotation\n");
bptr = aptr->lchild;
aptr->lchild = bptr->rchild;
bptr->rchild = aptr;
pptr->rchild = bptr->lchild;
bptr->lchild = pptr;
if(bptr->balance == -1)
pptr->balance = 1;
else
pptr->balance = 0;
if(bptr->balance == 1)
aptr->balance = -1;
else
aptr->balance = 0;
bptr->balance=0;
pptr = bptr;
}/*End of else*/
*ht_inc = FALSE;
}/*End of switch */
}/*End of if*/
}/*End of if*/
return(pptr);
}/*End of insert()*/
display(struct node *ptr,int level)
{
```

```
int i;
if ( ptr!=NULL )
{
display(ptr->rchild, level+1);
printf("\n");
for (i = 0; i < level; i++)
printf(" ");
printf("%d", ptr->info);
display(ptr->lchild, level+1);
}/*End of if*/
}/*End of display()*/
inorder(struct node *ptr)
{
if(ptr!=NULL)
{
inorder(ptr->lchild);
printf("%d ",ptr->info);
inorder(ptr->rchild);
}}/*End of inorder()*/
```

## OUTPUT



"E:\DESKTOP\DS LAB CS8381\avltree\bin\Debug\avltree.exe"

```
3.Quit
Enter your choice : 2
Tree is :

  56
 34
   12
  9
   2

Inorder Traversal is: 2 9 12 34 56
1.Insert
2.Display
3.Quit
Enter your choice : 3
```

## RESULT

Thus the ‘C’ program to implement an AVL trees . Produce its pre-Sequence, In-Sequence, and Post-Sequence traversals

# EXNO:8 IMPLEMENTATION OF PRIORITY QUEUE USING HEAPS

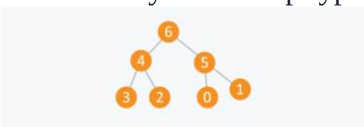
## AIM:

To write a C program to implement Priority Queue using Binary Heaps.

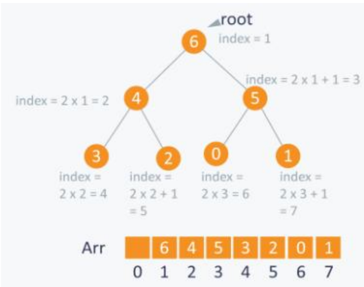
## DESCRIPTION:

A heap is a tree-based data structure in which all the nodes of the tree are in a specific order.

For example, if X is the parent node of Y, then the value of X follows a specific order with respect to the value of Y and the same order will be followed across the tree.
The maximum number of children of a node in a heap depends on the type of heap. However, in the more commonly-used heap type, there are at most 2 children of a node and it's known as a Binary heap




An array can be used to simulate a tree in the following way. If we are storing one element at index i in array Arr, then its parent will be stored at index i/2 (unless its a root, as root has no parent) and can be accessed by Arr[i/2], and its left child can be accessed by Arr[2∗i] and its right child can be accessed by Arr[2∗i+1]. Index of root will be 1 in an array.




## ALGORITHM:

1. Initialize all necessary variables and functions.
2. Read the choices.
3. For insertion, read the element to be inserted.
4. If root is NULL, assign the given element as root.
5. If the element is equal to the root, print “Duplicate value”.
6. Else if element value is less than the root value, insert element at the left of the root.
7. Else insert right side of the root.

8. For deletion, get the priority for maximum or minimum.

9. If maximum, it deletes the root and rearranges the tree.

10. If minimum, it deletes the leaf.

11. End of the program

## PROGRAM

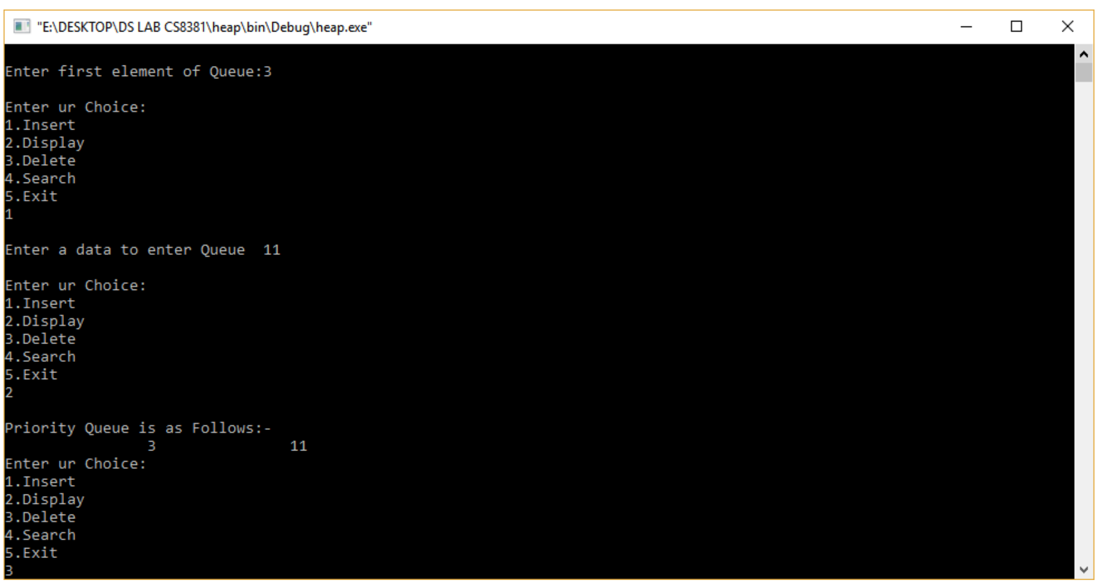
```
#include<stdio.h>
#include<conio.h>
#include <stdlib.h>
enum {FALSE=0,TRUE=-1};
struct Node
    {
      struct Node *Previous;
      int Data;
      struct Node *Next;
    }Current;
    struct Node *head;
    struct Node *ptr;
    static int NumOfNodes;
    int PriorityQueue(void);
    int Maximum(void);
    int Minimum(void);
    void Insert(int);
    int Delete(int);
    void Display(void);
    int Search (int);
    void main()
{
    int choice;
    int DT;
    PriorityQueue();

while(1)
      {
      printf("\nEnter ur Choice:");
      printf("\n1.Insert\n2.Display\n3.Delete\n4.Search\n5.Exit\n");
      scanf("%d",&choice);
      switch(choice)
      {
      case 1:
        printf("\nEnter a data to enter Queue");
        scanf("%d",&DT);
        Insert(DT);
          break;
      case 2:
          Display();
          break;
      case 3:
          {
            int choice,DataDel;
            printf("\nEnter ur choice:");
            printf("\n1.Maximum Priority queue\n2.Minimum priority Queue\n");
            scanf("%d",&choice);
            switch(choice)
            {
```

```
            case 1:
               DataDel=Maximum();
               Delete(DataDel);
               printf("\n%d is deleted\n",DataDel);
                break;
            case 2:


             DataDel=Minimum();
               Delete(DataDel);
               printf("\n%d is deleted\n",DataDel);
                break;
            default:
            printf("\nSorry Not a correct Choice\n");
            }
          }
          break;
      case 4:
          printf("\nEnter a data to Search in Queue:");
          scanf("%d",&DT);
          if(Search(DT)!=FALSE)
          printf("\n %d is present in queue",DT);
          else
          printf("\n%d is not present in queue",DT);
          break;
      case 5:
          exit(0);
      default:
      printf("\nCannot process ur choice\n");
      } }}
int PriorityQueue(void)
{
    Current.Previous=NULL;
    printf("\nEnter first element of Queue:");
    scanf("%d",&Current.Data);
    Current.Next=NULL;
    head=&Current;
    ptr=head;
    NumOfNodes++;
    return;
}
int Maximum(void)
{
    int Temp;
    ptr=head;
    Temp=ptr->Data;

    while(ptr->Next!=NULL)
    {
      if(ptr->Data>Temp)
          Temp=ptr->Data;
      ptr=ptr->Next;
    }
    if(ptr->Next==NULL && ptr->Data>Temp)
      Temp=ptr->Data;
    return(Temp);
}
int Minimum(void)
{
    int Temp;
    ptr=head;
    Temp=ptr->Data;
    while(ptr->Next!=NULL)
```

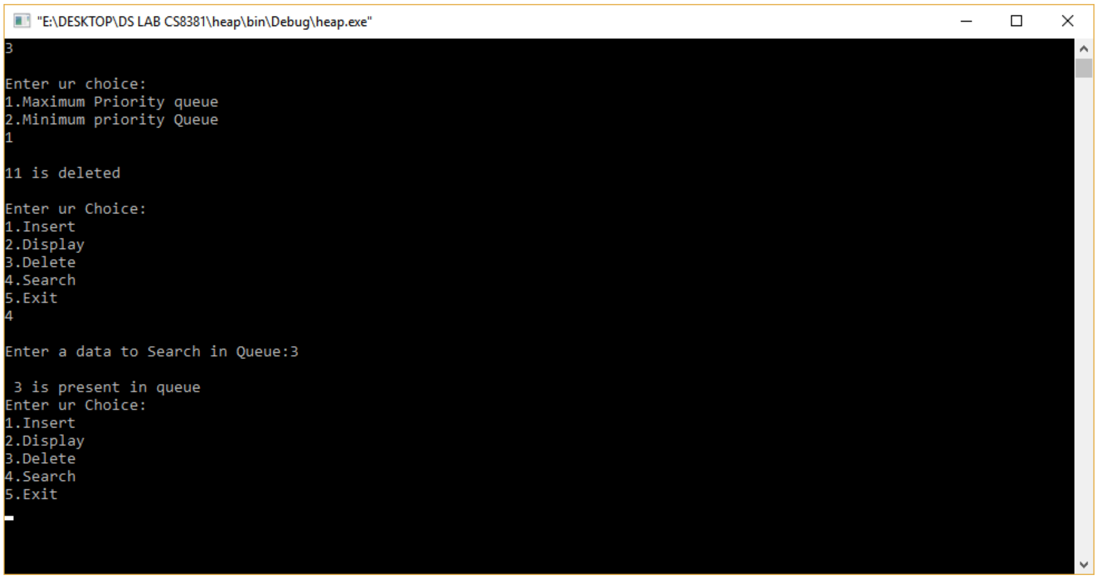
```
 {
      if(ptr->Data<Temp)
          Temp=ptr->Data;

      ptr=ptr->Next;
    }
    if(ptr->Next==NULL && ptr->Data<Temp)
      Temp=ptr->Data;
return(Temp);
}
void Insert(int DT)
{


       struct Node *newnode;
    newnode=(struct Node *)malloc(sizeof(struct Node));
    newnode->Next=NULL;
    newnode->Data=DT;
    while(ptr->Next!=NULL)
      ptr=ptr->Next;
    if(ptr->Next==NULL)
    {
      newnode->Next=ptr->Next;
      ptr->Next=newnode;
    }
    NumOfNodes++;
}

int Delete(int DataDel)
{
    struct Node *mynode,*temp;
    ptr=head;
    if(ptr->Data==DataDel)
    {
      temp=ptr;
      ptr=ptr->Next;
       ptr->Previous=NULL;
      head=ptr;
      NumOfNodes--;
      return(TRUE);
    }
    else
    {
      while(ptr->Next->Next!=NULL)
      {
         if(ptr->Next->Data==DataDel)
          {
            mynode=ptr;
            temp=ptr->Next;
            mynode->Next=mynode->Next->Next;
            mynode->Next->Previous=ptr;
            free(temp);
            NumOfNodes--;
            return(TRUE);
          }
          ptr=ptr->Next;
      }
      if(ptr->Next->Next==NULL && ptr->Next->Data==DataDel)
      {
          temp=ptr->Next;
          free(temp);
          ptr->Next=NULL;
          NumOfNodes--;
```

```
                return(TRUE);
            }
        }

return(FALSE);
}
int Search(int DataSearch)
{
    ptr=head;
    while(ptr->Next!=NULL)
    {


      if(ptr->Data==DataSearch)
          return ptr->Data;
      ptr=ptr->Next;
    }

      if(ptr->Next==NULL && ptr->Data==DataSearch)
      return ptr->Data;
    return(FALSE);
}
void Display(void)
{
    ptr=head;
    printf("\nPriority Queue is as Follows:-\n");
    while(ptr!=NULL)
    {
      printf("\t\t%d",ptr->Data);
      ptr=ptr->Next;
    }
}
```

## <u>OUTPUT</u>



```
"E:\DESKTOP\DS LAB CS8381\heap\bin\Debug\heap.exe"

Enter first element of Queue:3

Enter ur Choice:
1.Insert
2.Display
3.Delete
4.Search
5.Exit
1

Enter a data to enter Queue  11

Enter ur Choice:
1.Insert
2.Display
3.Delete
4.Search
5.Exit
2

Priority Queue is as Follows:-
                3               11
Enter ur Choice:
1.Insert
2.Display
3.Delete
4.Search
5.Exit
3
```



```
"E:\DESKTOP\DS LAB CS8381\heap\bin\Debug\heap.exe"
3

Enter ur choice:
1.Maximum Priority queue
2.Minimum priority Queue
1

11 is deleted

Enter ur Choice:
1.Insert
2.Display
3.Delete
4.Search
5.Exit
4

Enter a data to Search in Queue:3

 3 is present in queue
Enter ur Choice:
1.Insert
2.Display
3.Delete
4.Search
5.Exit
```

**RESULT:**

Thus the Priority Queue using Binary Heap is implemented and the result is verified successfully.

## Ex:10.A GRAPH REPRESENTATIONS

**AIM**:

To write a C program implement adjacent matrix and adjacency list .

**DESCRIPTION:**

Graph is a data structure that sconsists of following two components:

**1.** A finite set of vertices also called as nodes.

**2.** A finite set of ordered pair of the form (u, v) called as edge. The pair is ordered because (u, v) is not same as (v, u) in case of a directed graph(di-graph). The pair of the form (u, v) indicates that there is an edge from vertex u to vertex v. The edges may contain weight/value/cost.

Following two are the most commonly used representations of a graph.
**1.** Adjacency Matrix
**2.** Adjacency List
There are other representations also like, Incidence Matrix and Incidence List. The choice of the graph representation is situation specific. It totally depends on the type of operations to be performed and ease of use.

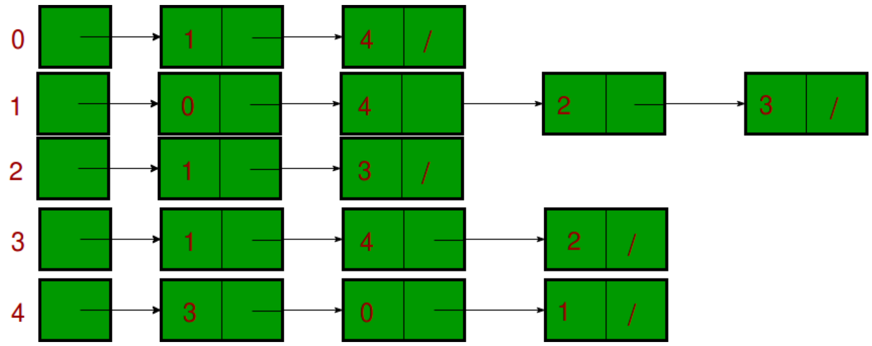
**Adjacency Matrix:**

Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph. Let the 2D array be adj[][], a slot adj[i][j] = 1 indicates that there is an edge from vertex i to vertex j. Adjacency matrix for undirected graph is always symmetric. Adjacency Matrix is also used to represent weighted graphs. If adj[i][j] = w, then there is an edge from vertex i to vertex j with weigh t w'.
The adjacency matrix for the above example graph is:

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 2 | 0 | 1 | 0 | 1 | 0 |
| 3 | 0 | 1 | 1 | 0 | 1 |
| 4 | 1 | 1 | 0 | 1 | 0 |

**Adjacency List:**

An array of linked lists is used. Size of the array is equal to the number of vertices. Let the array be array[]. An entry array[i] represents the linked list of vertices adjacent to the ***i***th vertex. This representation can also be used to represent a weighted graph. The weights of edges can be stored in nodes of linked lists. Following is adjacency list representation of the above graph..




## ALGORITHM:

1. Create a graph with getting no. of vertices and no. of edges

2. Implement adjacency matrix

3. Implement adjacency list

4. Close the program

## PROGRAM

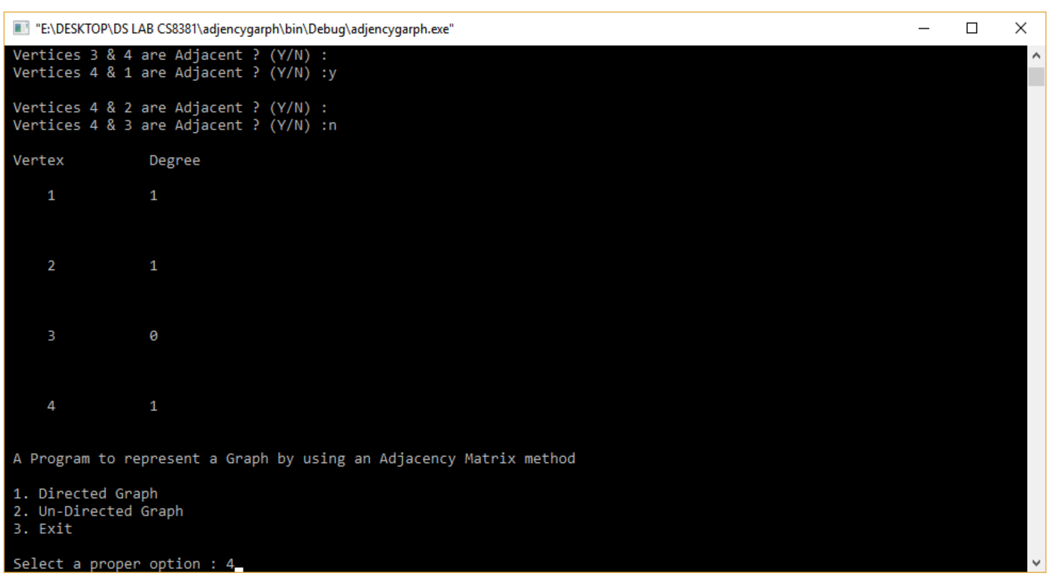
```
#include <stdio.h>

#include <stdlib.h>

void main()

{ int option;

   do

   {printf("\n A Program to represent a Graph by using an ");

      printf("Adjacency Matrix method \n ");

      printf("\n 1. Directed Graph ");

      printf("\n 2. Un-Directed Graph ");

      printf("\n 3. Exit ");

      printf("\n\n Select a proper option : ");

      scanf("%d", &option);

      switch(option)

      {
```

```
            case 1 : dir_graph();
                     break;
            case 2 : undir_graph();
                     break;
            case 3 : exit(0);
      } // switch
    }while(1);
}
int dir_graph()
{
    int adj_mat[50][50];
    int n;
    int in_deg, out_deg, i, j;
    printf("\n How Many Vertices ? : ");
    scanf("%d", &n);
    read_graph(adj_mat, n);
    printf("\n Vertex \t In_Degree \t Out_Degree \t Total_Degree ");
    for (i = 1; i <= n ; i++ )
    {
        in_deg = out_deg = 0;
      for ( j = 1 ; j <= n ; j++ )
      {
            if ( adj_mat[j][i] == 1 )
                in_deg++;}
        for ( j = 1 ; j <= n ; j++ )
            if (adj_mat[i][j] == 1 )
```

```
                out_deg++;

            printf("\n\n
%5d\t\t\t%d\t\t%d\t\t%d\n\n",i,in_deg,out_deg,in_deg+out_deg);

    }return;}

int undir_graph()

{

    int adj_mat[50][50];

    int deg, i, j, n;

    printf("\n How Many Vertices ? : ");

    scanf("%d", &n);

    read_graph(adj_mat, n);


      printf("\n Vertex \t Degree ");

    for ( i = 1 ; i <= n ; i++ )

    {

        deg = 0;

        for ( j = 1 ; j <= n ; j++ )

            if ( adj_mat[i][j] == 1)

                deg++;

        printf("\n\n %5d \t\t %d\n\n", i, deg);

    }

    return;}

int read_graph ( int adj_mat[50][50], int n )

{

    int i, j;

    char reply;

    for ( i = 1 ; i <= n ; i++ )
```

```
{
        for ( j = 1 ; j <= n ; j++ )
        {
            if ( i == j )
            {
                adj_mat[i][j] = 0;
            continue;
            }
            printf("\n Vertices %d & %d are Adjacent ? (Y/N) :",i,j);
            scanf("%c", &reply);
            if ( reply == 'y' || reply == 'Y' )
                adj_mat[i][j] = 1;
            else
                adj_mat[i][j] = 0;
      }}
    return;}
```

**OUTPUT**



**RESULT:**

Thus the C program implemented adjacent matrix and adjacency list

**EX.NO.10.B** Implement DFS and BFS graph traversal

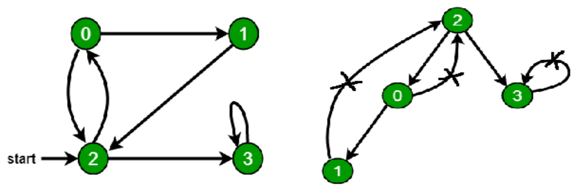
**Aim**:

To write a C program implement DFS and BFS graph traversal.

**DESCRIPTION:**

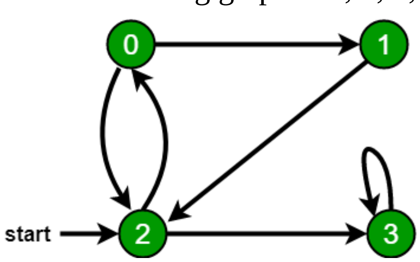
Depth First Traversal (or Search) for a graph is similar to Depth First Traversal of a tree. The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array.

For example, in the following graph, we start traversal from vertex 2. When we come to vertex 0, we look for all adjacent vertices of it. 2 is also an adjacent vertex of 0. If we don't mark visited vertices, then 2 will be processed again and it will become a non-terminating process. A Depth First Traversal of the following graph is 2, 0, 1, 3.




Breadth First Search or BFS for a Graph

Breadth First Traversal (or Search) for a graph is similar to Breadth First Traversal of a tree (See method 2 of this post). The only catch here is, unlike trees, graphs may contain cycles, so we may come to the same node again. To avoid processing a node more than once, we use a boolean visited array. For simplicity, it is assumed that all vertices are reachable from the starting vertex.
For example, in the following graph, we start traversal from vertex 2. When we come to vertex 0, we look for all adjacent vertices of it. 2 is also an adjacent vertex of 0. If we don't mark visited vertices, then 2 will be processed again and it will become a non-terminating process. A Breadth First Traversal of the following graph is 2, 0, 3, 1.




**ALGORITHM:**

**DFS**

1. Define a Stack of size total number of vertices in the graph.
2. Select any vertex as starting point for traversal. Visit that vertex and push it on to the Stack.
3. Visit any one of the adjacent vertex of the verex which is at top of the stack which is not visited and push it on to the stack.
4. Repeat step 3 until there are no new vertex to be visit from the vertex on top of the stack.
5. When there is no new vertex to be visit then use back tracking and pop one vertex from the stack.
6. Repeat steps 3, 4 and 5 until stack becomes Empty.

7. When stack becomes Empty, then produce final spanning tree by removing unused edges from the graph

**BFS**

1. Define a Queue of size total number of vertices in the graph.
2. Select any vertex as starting point for traversal. Visit that vertex and insert it into the Queue.
3. Visit all the adjacent vertices of the verex which is at front of the Queue which is not visited and insert them into the Queue.
4. When there is no new vertex to be visit from the vertex at front of the Queue then delete that vertex from the Queue.
5. Repeat step 3 and 4 until queue becomes empty.
6. When queue becomes Empty, then produce final spanning tree by removing unused edges from the graph

## PROGRAM

```
include<stdio.h>

int q[20],top=-1,front=-1,rear=-1,a[20][20],vis[20],stack[20];

int delete();

void add(int item);

void bfs(int s,int n);

void dfs(int s,int n);

void push(int item);

int pop();


void main()

{

int n,i,s,ch,j;

char c,dummy;

printf("ENTER THE NUMBER VERTICES ");

scanf("%d",&n);

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

printf("ENTER 1 IF %d HAS A NODE WITH %d ELSE 0 ",i,j);

scanf("%d",&a[i][j]);

}

}
```

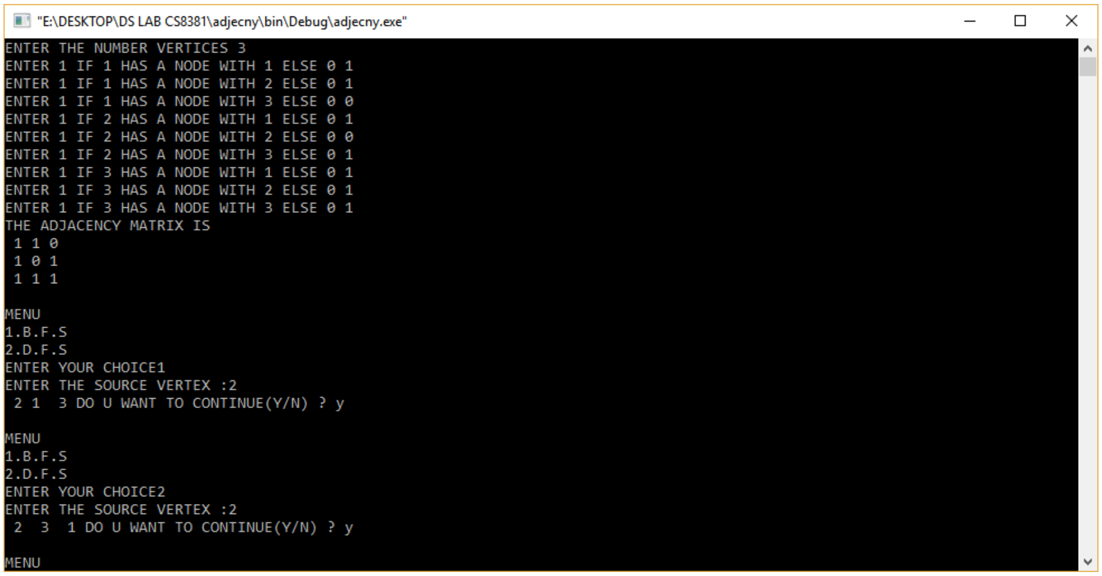
```
printf("THE ADJACENCY MATRIX IS\n");
for(i=1;i<=n;i++)
{
for(j=1;j<=n;j++)
{
printf(" %d",a[i][j]);
}
printf("\n");
}

do
{
for(i=1;i<=n;i++)
vis[i]=0;
printf("\nMENU");
printf("\n1.B.F.S");
printf("\n2.D.F.S");
printf("\nENTER YOUR CHOICE");
scanf("%d",&ch);
printf("ENTER THE SOURCE VERTEX :");
scanf("%d",&s);

switch(ch)
{
case 1:bfs(s,n);
break;
case 2:
dfs(s,n);
break;
}
printf("DO U WANT TO CONTINUE(Y/N) ? ");
scanf("%c",&dummy);
scanf("%c",&c);
```

```
}while((c=='y')||(c=='Y'));
}
//**************BFS(breadth-first search) code**************//
void bfs(int s,int n)
{
int p,i;
add(s);
vis[s]=1;
p=delete();
if(p!=0)
printf(" %d",p);
while(p!=0)
{
for(i=1;i<=n;i++)
if((a[p][i]!=0)&&(vis[i]==0))
{
add(i);
vis[i]=1;
}
p=delete();
if(p!=0)
printf(" %d ",p);
}
for(i=1;i<=n;i++)
if(vis[i]==0)
bfs(i,n);
}
void add(int item)
{
if(rear==19)
printf("QUEUE FULL");
else
{
if(rear==-1)
```

```
{
q[++rear]=item;
front++;
}
else
q[++rear]=item;
}
}
int delete()
{
int k;
if((front>rear)||(front==-1))
return(0);
else
{
k=q[front++];
return(k);
}
}
//***************DFS(depth-first search) code******************//
void dfs(int s,int n)
{
int i,k;
push(s);
vis[s]=1;
k=pop();
if(k!=0)
printf(" %d ",k);
while(k!=0)
{
for(i=1;i<=n;i++)
if((a[k][i]!=0)&&(vis[i]==0))
{push(i);
vis[i]=1;
```

```
}
k=pop();
if(k!=0)
printf(" %d ",k);
}
for(i=1;i<=n;i++)
if(vis[i]==0)
dfs(i,n);
}
void push(int item)
{
if(top==19)
printf("Stack overflow ");
else
stack[++top]=item;
}
int pop()
{
int k;
if(top==-1)
return(0);
else
{
k=stack[top--];
return(k);
}
}
```

## OUTPUT



```
"E:\DESKTOP\DS LAB CS8381\adjecny\bin\Debug\adjecny.exe"
ENTER THE NUMBER VERTICES 3
ENTER 1 IF 1 HAS A NODE WITH 1 ELSE 0 1
ENTER 1 IF 1 HAS A NODE WITH 2 ELSE 0 1
ENTER 1 IF 1 HAS A NODE WITH 3 ELSE 0 0
ENTER 1 IF 2 HAS A NODE WITH 1 ELSE 0 1
ENTER 1 IF 2 HAS A NODE WITH 2 ELSE 0 0
ENTER 1 IF 2 HAS A NODE WITH 3 ELSE 0 1
ENTER 1 IF 3 HAS A NODE WITH 1 ELSE 0 1
ENTER 1 IF 3 HAS A NODE WITH 2 ELSE 0 1
ENTER 1 IF 3 HAS A NODE WITH 3 ELSE 0 1
THE ADJACENCY MATRIX IS
 1 1 0
 1 0 1
 1 1 1

MENU
1.B.F.S
2.D.F.S
ENTER YOUR CHOICE1
ENTER THE SOURCE VERTEX :2
 2 1  3 DO U WANT TO CONTINUE(Y/N) ? y

MENU
1.B.F.S
2.D.F.S
ENTER YOUR CHOICE2
ENTER THE SOURCE VERTEX :2
 2  3  1 DO U WANT TO CONTINUE(Y/N) ? y

MENU
```

## RESULT:

Thus the C program implemented DFS and BFS graph traversal.

## Ex:11.A. IMPLEMENTATION OF SEARCHING ALGORITHMS LINEAR SEARCH AND BINARY SEARCH

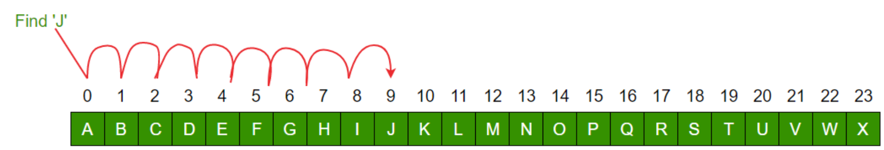
**AIM:**

To write a C Program to implement different searching techniques – Linear and Binary search.

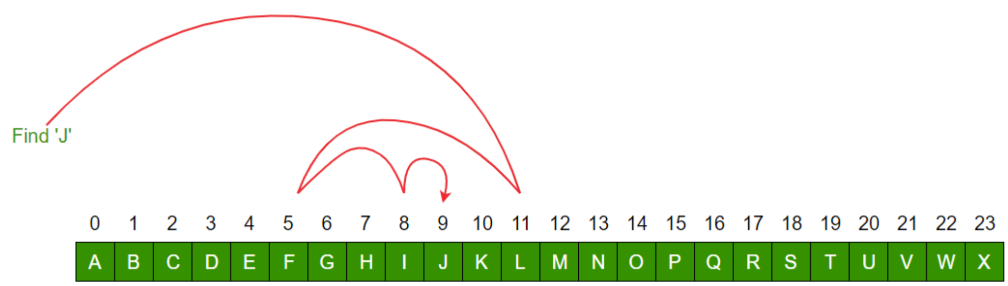
**DESCRIPTION:**

Binary search however, cut down your search to half as soon as you find middle of a sorted list.The middle element is looked to check if it is greater than or less than the value to be searched.Accordingly, search is done to either half of the given list

**Linear Search to find the element "J" in a given sorted list from A-X**




**Binary Search to find the element "J" in a given sorted list from A-X**




**ALGORITHM:**

**Linear Search**:

1. Read the search element from the user
2. Compare, the search element with the first element in the list.
3. If both are matching, then display "Given element found!!!" and terminate the function
4. If both are not matching, then compare search element with the next element in the list.
5. Repeat steps 3 and 4 until the search element is compared with the last element in the list.
6. If the last element in the list is also doesn't match, then display "Element not found!!!" and terminate the function.

**Binary search is implemented using following steps...**

1. Read the search element from the user
2. Find the middle element in the sorted list
3. Compare, the search element with the middle element in the sorted list.

4. If both are matching, then display "Given element found!!!" and terminate the function

5. If both are not matching, then check whether the search element is smaller or larger than middle element.

6. If the search element is smaller than middle element, then repeat steps 2, 3, 4 and 5 for the left sublist of the middle element.

7. If the search element is larger than middle element, then repeat steps 2, 3, 4 and 5 for the right sublist of the middle element.

8. Repeat the same process until we find the search element in the list or until sublist contains only one element.

9. If that element also doesn't match with the search element, then display "Element not found in the list!!!" and terminate the function.

**PROGRAM**

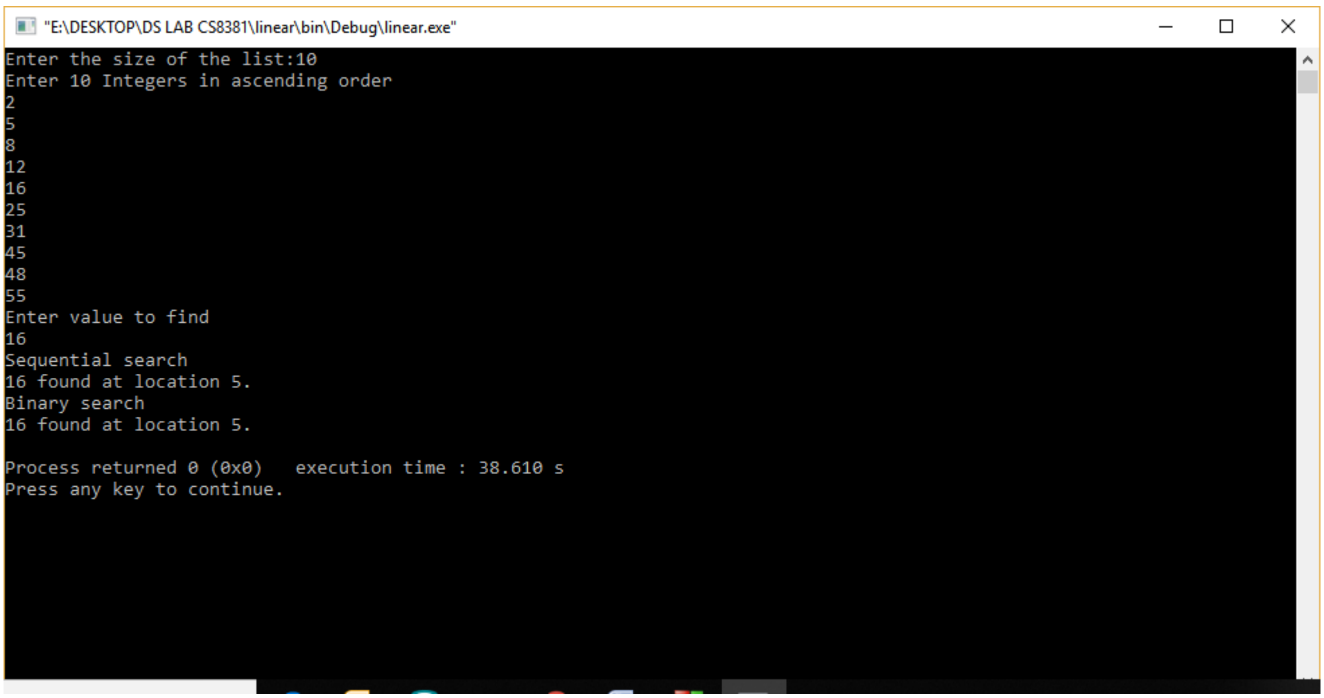
```
#include <stdio.h>
void sequential_search(int array[], int size, int n)
{
    int i;
    for (i = 0; i < size; i++)
    {
        if (array[i] == n)
        {
            printf("%d found at location %d.\n", n, i+1);
            break;
        }
    }
    if (i == size)
        printf("Not found! %d is not present in the list.\n", n);
}

void binary_search(int array[], int size, int n)
{
    int i, first, last, middle;
    first = 0;
    last = size - 1;
    middle = (first+last) / 2;

    while (first <= last) {
        if (array[middle] < n)
            first = middle + 1;
        else if (array[middle] == n) {
            printf("%d found at location %d.\n", n, middle+1);
           break;
        }
        else
            last = middle - 1;

        middle = (first + last) / 2;
    }
    if ( first > last )
        printf("Not found! %d is not present in the list.\n", n);
```

}



```
int main()
{
    int a[200], i, j, n, size;
    printf("Enter the size of the list:");
    scanf("%d", &size);
    printf("Enter %d Integers in ascending order\n", size);
    for (i = 0; i < size; i++)
        scanf("%d", &a[i]);
    printf("Enter value to find\n");
    scanf("%d", &n);
    printf("Sequential search\n");
    sequential_search(a, size, n);
    printf("Binary search\n");
    binary_search(a, size, n);
    return 0;
}
```

**OUTPUT**




**RESULT**

Thus the C Program to implement different searching techniques – Linear and Binary search

**Ex. No:11.B.1** **BUBBLE SORT**

**AIM:**

To write a C program to implement the concept of bubble sort

**DESCRIPTION:**

- Bubble sort is one of the simplest internal sorting algorithms.
- Bubble sort works by comparing two consecutive elements and the largest elementamong these two bubbles towards right at the end of the first pass the largest element gets sorted and placed at the end of the sorted list.
- This process is repeated for all pairs of elements until it moves the largest element to the end of the list in that iteration.
- Bubble sort consists of (n-1) passes, where n is the number of elements to be sorted.
- In $1^{st}$ pass the largest element will be placed in the $n^{th}$ position.
- In $2^{nd}$ pass the second largest element will be placed in the $(n-1)^{th}$ position.
- In $(n-1)^{th}$ pass only the first two elements are compared.

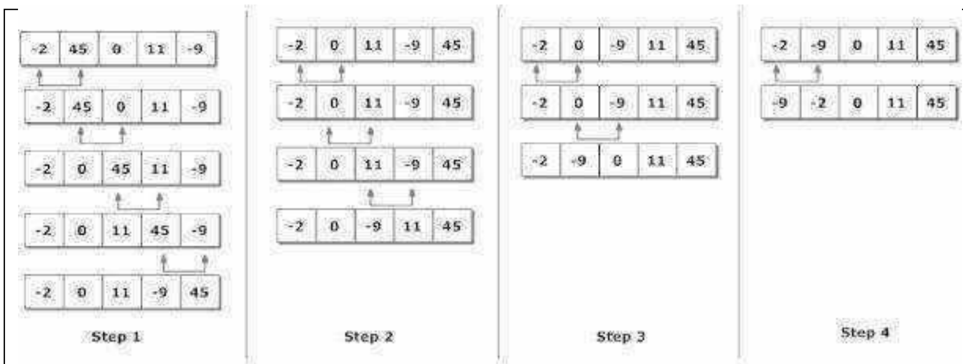


Figure: Working of Bubble sort algorithm

**ALGORITHM:**

1: Start.
2: Repeat Steps 3 and 4 for i=1 to 10
3: Set j=1
4: Repeat while j<=n

(A) if a[i] < a[j]

Then interchange a[i] and a[j]

[End of if]

(B) Set j = j+1
[End of Inner Loop]

[End of Step 1 Outer Loop]

5: Stop.

**PROGRAM:**

```
#include<stdio.h>

#include<conio.h>

void main(){

int n, i, j, temp , a[100];

printf("Enter the total integers you want to enter (make it less than 100):\n");

scanf("%d",&n);

printf("Enter the %d integer array elements:\n",n);

for(i=0;i<n;i++){

scanf("%d",&a[i]);

}

for(i=0;i<n-1;i++){

for(j=0;j<n-i-1;j++){

if(a[j+1]<a[j]){

temp = a[j];

a[j] = a[j+1];

a[j+1] = temp;

}}}

printf("The sorted numbers are:");

for(i=0;i<n;i++){

printf("%3d",a[i]);

}getch();

}
```

**OUTPUT**

```
"E:\DESKTOP\DS LAB CS8381\bubble\bin\Debug\bubble.exe"
Enter the total integers you want to enter (make it less than 100):
5
Enter the 5 integer array elements:
99
87
100
54
150
The sorted numbers are: 54 87 99100150
Process returned 0 (0x0)   execution time : 29.749 s
Press any key to continue.
```

**RESULT:**

Thus a C program for the concept of bubble sort was implemented successfull

**Ex. No:11.B.2 MERGE SORT**

**AIM:**

To write a C program to implement the concept of merge sort.

**DESCRIPTION:**

Merge sort is a sorting algorithm that uses the divide, conquer, and combine algorithmic paradigm.

Divide means partitioning the n-element array to be sorted into two sub-arrays of n/2 elements.

If there are more elements in the array, divide A into two sub-arrays, A1 and A2, each containing about half of the elements of A.

Conquer means sorting the two sub-arrays recursively using merge sort.

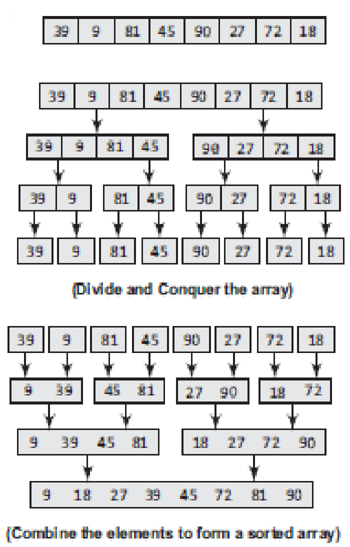
Combine means merging the two sorted sub-arrays of size n/2 to produce the sorted array of n elements.

The basic steps of a merge sort algorithm are as follows:

a. If the array is of length 0 or 1, then it is already sorted.

b. Otherwise, divide the unsorted array into two sub-arrays of about half the size. Use merge sort algorithm recursively to sort each sub-array.

Merge the two sub-arrays to form a single sorted list



**ALGORITHM:**

1: Start.

2: First you divide the number of elements by 2 and seperate them as two.

3: Divide those two which are divided by 2.

4: Divide them until you get a single element.

5: Start comparing the starting two pair of elements with each other and place them in ascending order.

6: When you combine them compare them so that you make sure they are sorted.

7: When all the elements are compared the array will be surely sorted in an ascending order.

8: Stop.

**PROGRAM:**

```
#include<stdio.h>

#include<conio.h>

void merge(int [],int ,int ,int );

void part(int [],int ,int );

void main(){

int arr[30];

int i,size;

printf("\n\t------- Merge sorting method -------\n\n");

printf("Enter total no. of elements : ");

scanf("%d",&size);

for(i=0; i<size; i++){

printf("Enter %d element : ",i+1);

scanf("%d",&arr[i]);

}

part(arr,0,size-1);

printf("\n\t------- Merge sorted elements -------\n\n");

for(i=0; i<size; i++)

printf("%d ",arr[i]);

getch();

}

void part(int arr[],int min,int max){

int mid;

if(min<max){

mid=(min+max)/2;

part(arr,min,mid);

part(arr,mid+1,max);

merge(arr,min,mid,max);}}

void merge(int arr[],int min,int mid,int max){

int tmp[30];
```

```
int i,j,k,m;
j=min;
m=mid+1;
for(i=min; j<=mid && m<=max ; i++){
if(arr[j]<=arr[m]){
tmp[i]=arr[j];
j++;}
else{
tmp[i]=arr[m];
m++;
}}
if(j>mid){
for(k=m; k<=max; k++){
tmp[i]=arr[k];
i++;
}}
else{
for(k=j; k<=mid; k++){
tmp[i]=arr[k];
i++;
}}
for(k=min; k<=max; k++)
arr[k]=tmp[k]; }
```

## OUTPUT

```
"E:\DESKTOP\DS LAB CS8381\merge\bin\Debug\merge.exe"

        ------- Merge sorting method -------

Enter total no. of elements : 9
Enter 1 element : 100
Enter 2 element : 99
Enter 3 element : 75
Enter 4 element : 155
Enter 5 element : 11
Enter 6 element : 43
Enter 7 element : 13
Enter 8 element : 35
Enter 9 element : 48

        ------- Merge sorted elements -------

11 13 35 43 48 75 99 100 155
Process returned 0 (0x0)   execution time : 42.560 s
Press any key to continue.
```

## RESULT:

Thus a C program for the concept of merge sort was implemented successfully.

**Ex. No:11.B.3** **QUICK SORT**

**AIM:**

To write a C program to implement the concept of Quick sort.

**DESCRIPTION:**

Quick Sort, as the name suggests, sorts any list very quickly. Quick sort is not stable search, but it is very fast and requires very less additional space. It is based on the rule of Divide and Conquer (also called partition-exchange sort). This algorithm divides the list into three main parts:

- Elements less than the Pivot element
- Pivot element
- Elements greater than the pivot element

**ALGORITHM:**

1: Start.

2: Choose any element of the array to be the pivot.

3: Divide all other elements (except the pivot) into two partitions.

  - All elements less than the pivot must be in the first partition.
  - All elements greater than the pivot must be in the second partition.

4: Use recursion to sort both partitions.

5: Join the first sorted partition, the pivot, and the second sorted partition.

6: Stop

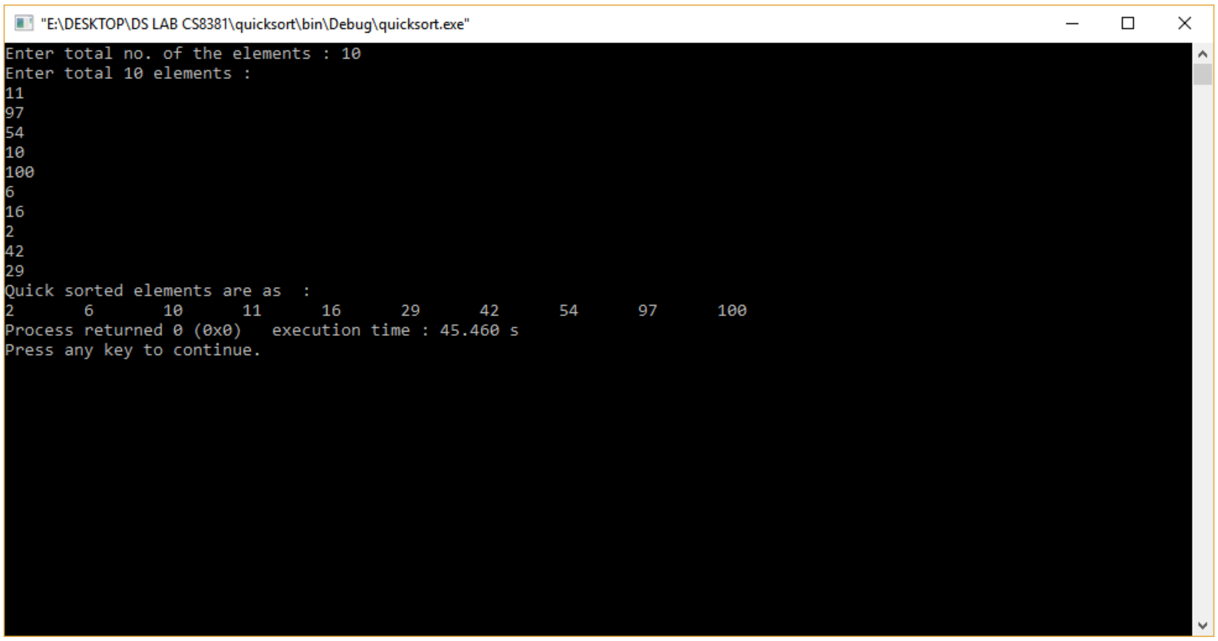
**PROGRAM:**

```
#include<stdio.h>

#include<conio.h>

void qsort(int arr[20], int fst, int last);

void main(){

int arr[30];

int i,size;
```

```
printf("Enter total no. of the elements : ");

scanf("%d",&size);

printf("Enter total %d elements : \n",size);

for(i=0; i<size; i++)

scanf("%d",&arr[i]);

qsort(arr,0,size-1);

printf("Quick sorted elements are as  : \n");

for(i=0; i<size; i++)

printf("%d\t",arr[i]);

 getch();}

void qsort(int arr[20], int fst, int last){

int i,j,pivot,tmp;

if(fst<last){

pivot=fst;

i=fst;

j=last;

while(i<j){

while(arr[i]<=arr[pivot] && i<last)

i++;

while(arr[j]>arr[pivot])

j--;

if(i<j){

tmp=arr[i];

arr[i]=arr[j];

arr[j]=tmp;}}

tmp=arr[pivot];

arr[pivot]=arr[j];
```

```
arr[j]=tmp;

qsort(arr,fst,j-1);

qsort(arr,j+1,last);

} }
```

## OUTPUT




## RESULT:

Thus the C program to implement the concept of Quick sort.

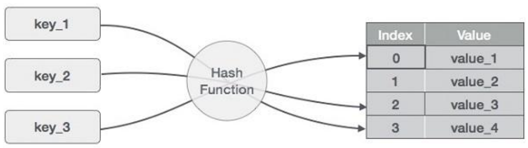
**Ex. No: 11** **HASHING TECHNIQUES**

**AIM**: To write a C program to implement hash table

## DESCRIPTION:

### Hashing

Hashing is a technique to convert a range of key values into a range of indexes of an array. We're going to use modulo operator to get a range of key values. Consider an example of hash table of size 20, and the following items are to be stored. Item are in the (key,value) format.




- (1,20)
- (2,70)
- (42,80)
- (4,25)
- (12,44)
- (14,32)
- (17,11)
- (13,78)
- (37,98)

| Sr.No. | Key | Hash | Array Index |
|---|---|---|---|
| 1 | 1 | 1 % 20 = 1 | 1 |
| 2 | 2 | 2 % 20 = 2 | 2 |
| 3 | 42 | 42 % 20 = 2 | 2 |
| 4 | 4 | 4 % 20 = 4 | 4 |
| 5 | 12 | 12 % 20 = 12 | 12 |
| 6 | 14 | 14 % 20 = 14 | 14 |
| 7 | 17 | 17 % 20 = 17 | 17 |
| 8 | 13 | 13 % 20 = 13 | 13 |
| 9 | 37 | 37 % 20 = 17 | 17 |

## ALGORITHM:

1. Create a structure, data (hash table item) with key and value as data.

2. Now create an array of structure, data of some certain size (10, in this case). But, the size of array must be immediately updated to a prime number just greater than initial array capacity (i.e 10, in this case).

3. A menu is displayed on the screen.

4. User must choose one option from four choices given in the menu

5. Perform all the operations

6. Stop the program

## PROGRAM

```
#include<stdio.h>

#include<stdlib.h>

struct data

{

      int key;

      int value;

};

struct data *array;

int capacity = 10;

int size = 0;

/* this function gives a unique hash code to the given key */

int hashcode(int key)

{

      return (key % capacity);

}

/* it returns prime number just greater than array capacity */

int get_prime(int n)

{

      if (n % 2 == 0)

        {
```

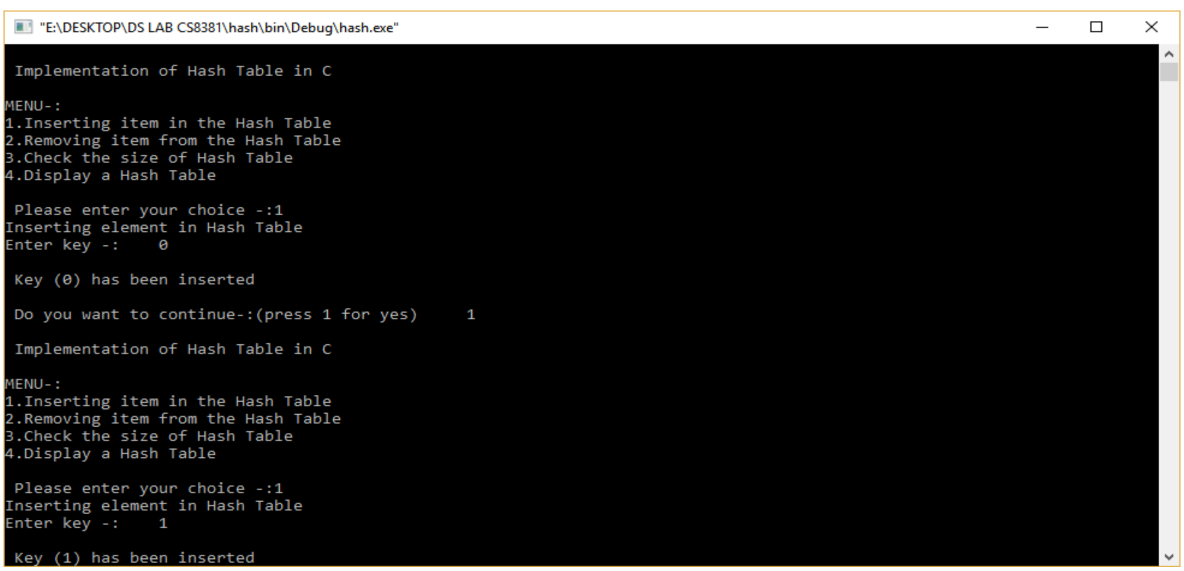
```
n++;
      }
      for (; !if_prime(n); n += 2);
      return n;
}
/* to check if given input (i.e n) is prime or not */
int if_prime(int n)
{
      int i;
      if ( n == 1  ||  n == 0)
        {
            return 0;
      }
      for (i = 2; i < n; i++)
        {
            if (n % i == 0)
                {
                  return 0;}}
      return 1;
}
void init_array()
{
      int i;
      capacity = get_prime(capacity);
      array = (struct data*) malloc(capacity * sizeof(struct data));
      for (i = 0; i < capacity; i++)
        {
            array[i].key = 0;
            array[i].value = 0;
      }}
/* to insert a key in the hash table */
void insert(int key)
{
      int index = hashcode(key);
```

```
if (array[index].value == 0)
        {
            /*  key not present, insert it  */
            array[index].key = key;
            array[index].value = 1;
            size++;
            printf("\n Key (%d) has been inserted \n", key);
      }
      else if(array[index].key == key)
        {
            /*  updating already existing key  */
      printf("\n Key (%d) already present, hence updating its value \n", key);
            array[index].value += 1;
      }
      else
        {
      /*  key cannot be insert as the index is already containing some other key*/
            printf("\n ELEMENT CANNOT BE INSERTED \n");
      }}
/* to remove a key from hash table */
void remove_element(int key)
{
      int index  = hashcode(key);
      if(array[index].value == 0)
        {
            printf("\n This key does not exist \n");
      }
      else {
            array[index].key = 0;
            array[index].value = 0;
            size--;
            printf("\n Key (%d) has been removed \n", key);}}
/* to display all the elements of a hash table */
void display()
```

```
{
      int i;for (i = 0; i < capacity; i++)
        {
            if (array[i].value == 0)
                {
                  printf("\n Array[%d] has no elements \n");
            }
            else
                {
printf("\n array[%d] has elements -:\n key(%d) and value(%d) \t", i, array[i].key,
array[i].value);
            }}}
int size_of_hashtable()
{
      return size;
}
void main()
{
      int choice, key, value, n, c;
      init_array();
      do {
            printf("\n Implementation of Hash Table in C \n\n");
            printf("MENU-:  \n1.Inserting item in the Hash Table"
                                   "\n2.Removing item from the Hash Table"
                               "\n3.Check the size of Hash Table"
                                   "\n4.Display a Hash Table"
                   "\n\n Please enter your choice -:");
            scanf("%d", &choice);
            switch(choice)
                {
            case 1:
                  printf("Inserting element in Hash Table\n");
                  printf("Enter key -:\t");
                  scanf("%d", &key);
```

```
insert(key);
                 break;
           case 2:
                printf("Deleting in Hash Table \n Enter the key to delete-:");
                 scanf("%d", &key);
                 remove_element(key);
                 break;
     case 3:
                 n = size_of_hashtable();
                 printf("Size of Hash Table is-:%d\n", n);
                 break;
           case 4:
                 display();
                 break;
           default:
                  printf("Wrong Input\n");}
           printf("\n Do you want to continue-:(press 1 for yes)\t");
           scanf("%d", &c);
     }while(c == 1);
     getch();}
```

## OUTPUT



```
"E:\DESKTOP\DS LAB CS8381\hash\bin\Debug\hash.exe"

 Implementation of Hash Table in C

MENU-:
1.Inserting item in the Hash Table
2.Removing item from the Hash Table
3.Check the size of Hash Table
4.Display a Hash Table

 Please enter your choice -:1
Inserting element in Hash Table
Enter key -:    0

 Key (0) has been inserted

 Do you want to continue-:(press 1 for yes)     1

 Implementation of Hash Table in C

MENU-:
1.Inserting item in the Hash Table
2.Removing item from the Hash Table
3.Check the size of Hash Table
4.Display a Hash Table

 Please enter your choice -:1
Inserting element in Hash Table
Enter key -:    1

 Key (1) has been inserted
```

```
"E:\DESKTOP\DS LAB CS8381\hash\bin\Debug\hash.exe"
1.Inserting item in the Hash Table
2.Removing item from the Hash Table
3.Check the size of Hash Table
4.Display a Hash Table

 Please enter your choice -:4

 array[0] has elements -:
 key(0) and value(1)
 array[1] has elements -:
 key(1) and value(1)
 Array[1] has no elements

 Array[1] has no elements

 Array[1] has no elements

 Array[1] has no elements

 Array[1] has no elements

 Array[1] has no elements

 Array[1] has no elements

 Array[1] has no elements

 Array[1] has no elements

 Do you want to continue-:(press 1 for yes)
```

**RESULT**: Thus the C program to implemented Hash Table.

# CS8381– DATA STRUCTURES LABORATORY

## VIVA VOCE INTERVIEW QUESTIONS AND ANSWERS

1) What is data structure?

Data structure refers to the way data is organized and manipulated. It seeks to find ways to make data access more efficient. When dealing with the data structure, we not only focus on one piece of data but the different set of data and how they can relate to one another in an organized manner.

2) Differentiate between file and structure storage structure.

The key difference between both the data structure is the memory area that is being accessed. When dealing with the structure that resides the main memory of the computer system, this is referred to as storage structure. When dealing with an auxiliary structure, we refer to it as file structures.

3) When is a binary search best applied?

A binary search is an algorithm that is best applied to search a list when the elements are already in order or sorted. The list is searched starting in the middle, such that if that middle value is not the target search key, it will check to see if it will continue the search on the lower half of the list or the higher half. The split and search will then continue in the same manner.

4) What is a linked list?

A linked list is a sequence of nodes in which each node is connected to the node following it. This forms a chain-like link for data storage.

5) How do you reference all the elements in a one-dimension array?

To reference all the elements in a one -dimension array, you need to use an indexed loop, So that, the counter runs from 0 to the array size minus one. In this manner, You can reference all the elements in sequence by using the loop counter as the array subscript.

6) In what areas do data structures are applied?

Data structures are essential in almost every aspect where data is involved. In general, algorithms that involve efficient data structure is applied in the following areas: numerical analysis, operating system, A.I., compiler design, database management, graphics, and statistical analysis, to name a few.

7) What is LIFO?

LIFO is a short form of Last In First Out. It refers how data is accessed, stored and retrieved. Using this scheme, data that was stored last should be the one to be extracted first. This also means that in order to gain access to the first data, all the other data that was stored before this first data must first be retrieved and extracted.

8 ) What is a queue?

A queue is a data structure that can simulate a list or stream of data. In this structure, new elements are inserted at one end, and existing elements are removed from the other end.

9) What are binary trees?

A binary tree is one type of data structure that has two nodes, a left node, and a right node. In programming, binary trees are an extension of the linked list structures.

10) Which data structures are applied when dealing with a recursive function?

Recursion, is a function that calls itself based on a terminating condition, makes use of the stack. Using LIFO, a call to a recursive function saves the return address so that it knows how to return to the calling function after the call terminates.

11) What is a stack?

A stack is a data structure in which only the top element can be accessed. As data is stored in the stack, each data is pushed downward, leaving the most recently added data on top.

12) Explain Binary Search Tree

A binary search tree stores data in such a way that they can be retrieved very efficiently. The left subtree contains nodes whose keys are less than the node's key value, while the right subtree contains nodes whose keys are greater than or equal to the node's key value. Moreover, both subtrees are also binary search trees.

13) What are multidimensional arrays?

Multidimensional arrays make use of multiple indexes to store data. It is useful when storing data that cannot be represented using single dimensional indexing, such as data representation in a board game, tables with data stored in more than one column.

14) Are linked lists considered linear or non-linear data structures?

It depends on where you intend to apply linked lists. If you based it on storage, a linked list is considered non-linear. On the other hand, if you based it on access strategies, then a linked list is considered linear.

15) How does dynamic memory allocation help in managing data?

Apart from being able to store simple structured data types, dynamic memory allocation can combine separately allocated structured blocks to form composite structures that expand and contract as needed.

16) What is FIFO?

FIFO stands for First-in, First-out, and is used to represent how data is accessed in a queue. Data has been inserted into the queue list the longest is the one that is removed first.

17) What is an ordered list?

An ordered list is a list in which each node's position in the list is determined by the value of its key component, so that the key values form an increasing sequence, as the list is traversed.

18) What is merge sort?

Merge sort, is a divide-and-conquer approach for sorting the data. In a sequence of data, adjacent ones are merged and sorted to create bigger sorted lists. These sorted lists are then merged again to form an even bigger sorted list, which continues until you have one single sorted list.

19) Differentiate NULL and VOID

Null is a value, whereas Void is a data type identifier. A variable that is given a Null value indicates an empty value. The void is used to identify pointers as having no initial size.

20) What is the primary advantage of a linked list?

A linked list is an ideal data structure because it can be modified easily. This means that editing a linked list works regardless of how many elements are in the list.

21) What is the difference between a PUSH and a POP?

Pushing and popping applies to the way data is stored and retrieved in a stack. A push denotes data being added to it, meaning data is being "pushed" into the stack. On the other hand, a pop denotes data retrieval, and in particular, refers to the topmost data being accessed.

22) What is a linear search?

A linear search refers to the way a target key is being searched in a sequential data structure. In this method, each element in the list is checked and compared against the target key. The process is repeated until found or if the end of the file has been reached.

23) How does variable declaration affect memory allocation?

The amount of memory to be allocated or reserved would depend on the data type of the variable being declared. For example, if a variable is declared to be of integer type, then 32 bits of memory storage will be reserved for that variable.

24) What is the advantage of the heap over a stack?

The heap is more flexible than the stack. That's because memory space for the heap can be dynamically allocated and de-allocated as needed. However, the memory of the heap can at times be slower when compared to that stack.

25) What is a postfix expression?

A postfix expression is an expression in which each operator follows its operands. The advantage of this form is that there is no need to group sub-expressions in parentheses or to consider operator precedence.

26) What is Data abstraction?

Data abstraction is a powerful tool for breaking down complex data problems into manageable chunks. This is applied by initially specifying the data objects involved and the operations to be performed on these data objects without being overly concerned with how the data objects will be represented and stored in memory.

27) How do you insert a new item in a binary search tree?

Assuming that the data to be inserted is a unique value (that is, not an existing entry in the tree), check first if the tree is empty. If it's empty, just insert the new item in the root node. If it's not empty, refer to the new item's key. If it's smaller than the root's key, insert it into the root's left subtree, otherwise, insert it into the root's right subtree.

28) How does a selection sort work for an array?

The selection sort is a fairly intuitive sorting algorithm, though not necessarily efficient. In this process, the smallest element is first located and switched with the element at subscript zero, thereby placing the smallest element in the first position.

The smallest element remaining in the subarray is then located next to subscripts 1 through n-1 and switched with the element at subscript 1, thereby placing the second smallest element in the second position. The steps are repeated in the same manner till the last element.

29) How do signed and unsigned numbers affect memory?

In the case of signed numbers, the first bit is used to indicate whether positive or negative, which leaves you with one bit short. With unsigned numbers, you have all bits available for that number. The effect is best seen in the number range (an unsigned 8-bit number has a range 0-255, while the 8-bit signed number has a range -128 to +127.

30) What is the minimum number of nodes that a binary tree can have?

A binary tree can have a minimum of zero nodes, which occurs when the nodes have NULL values. Furthermore, a binary tree can also have 1 or 2 nodes.

31) What are dynamic data structures?

Dynamic data structures are structures that expand and contract as a program runs. It provides a flexible means of manipulating data because it can adjust according to the size of the data.

32) In what data structures are pointers applied?

Pointers that are used in linked list have various applications in the data structure. Data structures that make use of this concept include the Stack, Queue, Linked List and Binary Tree.

33) Do all declaration statements result in a fixed reservation in memory?

Most declarations do, with the exemption of pointers. Pointer declaration does not allocate memory for data, but for the address of the pointer variable. Actual memory allocation for the data comes during run-time.

34) What are ARRAYs?

When dealing with arrays, data is stored and retrieved using an index that refers to the element number in the data sequence. This means that data can be accessed in any order. In programming, an array is declared as a variable having a number of indexed elements.

35) What is the minimum number of queues needed when implementing a priority queue?

The minimum number of queues needed in this case is two. One queue is intended for sorting priorities while the other queue is used for actual storage of data.

36) Which sorting algorithm is considered the fastest?

There are many types of sorting algorithms: quick sort, bubble sort, balloon sort, radix sort, merge sort, etc. Not one can be considered the fastest because each algorithm is designed for a particular data structure and data set. It would depend on the data set that you would want to sort.

37) Differentiate STACK from ARRAY.

Stack follows a LIFO pattern. It means that data access follows a sequence wherein the last data to be stored when the first one to be extracted. Arrays, on the other hand, does not follow a particular order and instead can be accessed by referring to the indexed element within the array.

38) Give a basic algorithm for searching a binary search tree.

if the tree is empty, then the target is not in the tree, end search
2. if the tree is not empty, the target is in the tree
3. check if the target is in the root item
4. if a target is not in the root item, check if a target is smaller than the root’s value
5. if a target is smaller than the root’s value, search the left subtree
6. else, search the right subtree

39) What is a dequeue?

A dequeue is a double-ended queue. This is a structure wherein elements can be inserted or removed from either end.

40) What is a bubble sort and how do you perform it?

A bubble sort is one sorting technique that can be applied to data structures such as an array. It works by comparing adjacent elements and exchanges their values if they are out of order. This method lets the smaller values “bubble” to the top of the list, while the larger value sinks to the bottom.

41) What are the parts of a linked list?

A linked list typically has two parts: the head and the tail. Between the head and tail lie the actual nodes. All these nodes are linked sequentially.

42) How does selection sort work?

Selection sort works by picking the smallest number from the list and placing it at the front. This process is repeated for the second position towards the end of the list. It is the simplest sort algorithm.

43) What is a graph?

A graph is one type of data structure that contains a set of ordered pairs. These ordered pairs are also referred to as edges or arcs and are used to connect nodes where data can be stored and retrieved.

44) Differentiate linear from a nonlinear data structure.

The linear data structure is a structure wherein data elements are adjacent to each other. Examples of linear data structure include arrays, linked lists, stacks, and queues. On the other hand, a non-linear data structure is a structure wherein each data element can connect to more than two adjacent data elements. Examples of nonlinear data structure include trees and graphs.

45) What is an AVL tree?

An AVL tree is a type of binary search tree that is always in a state of partially balanced. The balance is measured as a difference between the heights of the subtrees from the root. This self-balancing tree was known to be the first data structure to be designed as such.

46) What are doubly linked lists?

Doubly linked lists are a special type of linked list wherein traversal across the data elements can be done in both directions. This is made possible by having two links in every node, one that links to the next node and another one that connects to the previous node.

47) What is Huffman’s algorithm?

Huffman’s algorithm is used for creating extended binary trees that have minimum weighted path lengths from the given weights. It makes use of a table that contains the frequency of occurrence for each data element.

48) What is Fibonacci search?

Fibonacci search is a search algorithm that applies to a sorted array. It makes use of a divide-and-conquer approach that can significantly reduce the time needed in order to reach the target element.

49) Briefly explain recursive algorithm.

Recursive algorithm targets a problem by dividing it into smaller, manageable sub-problems. The output of one recursion after processing one sub-problem becomes the input to the next recursive process.

50) How do you search for a target key in a linked list?

To find the target key in a linked list, you have to apply sequential search. Each node is traversed and compared with the target key, and if it is different, then it follows the link to the next node. This traversal continues until either the target key is found or if the last node is reach

## CONTENT BEYOND THE SYLLABUS

EX NO:1 **SINGLY LINKED LIST IMPLEMENTATION**

**AIM:**

To implement singly linked list and performing insert, delete and search operations.

**ALGORITHM:**

1. Set a node to contain INFO and LINK fields.
2. Allot memory dynamically for a node and declare it as a header H.
3. To create a singly linked lists get the element N and allot memory for a node S1.
4. Set S1->INFO=N; and S1->LINK=NULL.
5. Repeat the above two steps for all the elements.
6. A node can be inserted at the front, in the middle or at the end of the list.
7. To insert a node X at the front check whether the list is empty, if not set X->LINK=H->LINK and H->LINK=X.
8. To insert a node X at the end travel till the end of the list and assign the last node's LINK value to X.
9. To insert a node X after the specified node Y, travel the list till the node Y is reached. Set X->LINK=Y->LINK and Y->LINK=X
10. A node can be deleted at the front, in the middle or at the end of the list.
11. To delete a node X at the front set H->LINK=H->LINK->LINK.
12. To delete a node X at the end travel the list till the end and assign the previous to last node's LINK value to be NULL.
13. To delete a node X after the specified node Y, travel the list till the node Y is reached Set Y->LINK= Y->LINK->LINK.
14. To search an element E traverse the list until E is found.

****** SINGLY LINKED LIST ******

## PROGRAM

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define TRUE 1
#define FALSE 0
typedef struct SLL
{
int data;
struct SLL *next;
}node;
node *create();
void main()
{
int choice,val;
char ans;
node *head;
void display(node *);
node *search(node *,int);
void insert(node *);
void delete(node **);
node *get_prev(node *,int);
head=NULL;
clrscr();
do
{
printf("\n Singly Linked List\n");
printf("\n 1.create\n2.Display\n3.search\n4.insert\n5.delete\n6.quit\n");
printf("\n Enter ur choice:");
scanf("%d",&choice);
switch(choice)
{
```

```
case 1:
head=create();
break;
case 2:display(head);
break;
case 3:printf("Enter the element to search");
scanf("%d",&val);
search(head,val);
break;
case 4:insert(head);
break;
case 5:delete(&head);
break;
case 6:exit(0);
efault:clrscr();
printf("Invalid choice, try again");
getch();
}
}
while(choice!=6);
}
node *create()
{
node *temp,*new,*head;
int val,flag;
char ans='y';
node *get_node();
temp=NULL;
flag=TRUE;
do
{
printf("\nEmter the element:");
scanf("%d",&val);
new=get_node();
if(new==NULL)
```

```
printf("\nMemory is not allocated");
new->data=val;
if(flag)
{head=new;
temp=head;
flag=FALSE;
}
else
{
temp->next=new;
temp=new;
}
printf("\n Do u want to enter more elements?");
ans=getche();
}while(ans=='y');
printf("\nThe singly linked list is created\n");
getch();
return head;
}
node *get_node()
{
node *temp;
temp=(node *)malloc(sizeof(node));
temp->next=NULL;
return temp;
}
void display(node *head)
{
node *temp;
temp=head;
if(temp==NULL)
{
printf("\nThe listis empty\n");
getch();
```

```
return;
}
while(temp!=NULL)
{
printf("%d->",temp->data);
temp=temp->next;
}
printf("NULL");
getch();
}
node* search(node *head,int key)
{
node *temp;
int found;
temp=head;
if(temp==NULL)
{
printf("\nThe listis empty\n");
getch();
return NULL;
}
found=FALSE;
while(temp!=NULL && !found)
{
if(temp->data!=key)
temp=temp->next;
else
found=TRUE;
}
if(found)
{
printf("\nThe element is present\n");
getch();
return temp;
```

```
}
else
{
printf("\nThe element is not found\n");
getch();
return NULL;
}
}
void insert(node *head)
{
node *temp,*new;
int val;
temp=head;
if(temp==NULL)
{
printf("\nInsertion is not possible\n");
getch();
return;
}
printf("\n Enter the element after which to insert:");
scanf("%d",&val);
temp=search(head,val);
if(temp!=NULL)
{
printf("Enter the element to insert:");
scanf("%d",&val);
new=(node*)malloc(sizeof(node));
if(new==NULL)
printf("memory is not allocated\n");
new->data=val;
new->next=NULL;
new->next=temp->next;
temp->next=new;
printf("\n The element is inserted");
```

```
getch();
}
}
node* get_prev(node *head,int val)
{
node *temp,*prev;
int flag;
temp=head;
if(temp==NULL)
return NULL;
flag=FALSE;
prev=NULL;
while(temp!=NULL && !flag)
{
if(temp->data!=val)
{
prev=temp;
temp=temp->next;
}
else
flag=TRUE;
}
if(flag)
return prev;
else
return NULL;
}
void delete(node **head)
{
node *temp,*prev;
int key;
temp=*head;
if(temp==NULL)
{
```

```
printf("\nThe list is empty\n");
getch();
return;
}
printf("\nEnter the element u want to delete:");
scanf("%d",&key);
temp=search(*head,key);
if(temp!=NULL)
{
prev=get_prev(*head,key);
if(prev!=NULL)
{
prev->next=temp->next;
free(temp);
}
else
{
*head=temp->next;
free(temp);
}
printf("\nThe element is deleted\n");
getch();
}
}
```

**OUTPUT:**

Singly Linked List

1.create

2.Display

3.search

4.insert

5.delete

6.quit

Enter ur choice:1

Enter the element:12

Do u want to enter more elements?y

Enter the element:11

Do u want to enter more elements?n

The singly linked list is created

Singly Linked List

1.create

2.Display

3.search

4.insert

5.delete

6.quit

Enter ur choice:2

12->11->NULL

Singly Linked List

1.Create

2.Display

3.search

4.insert

5.delete

6.quit

Enter ur choice:3

Enter the element to search 11

The element is present

Singly Linked List

1.create

2.Display

3.search

4.insert

5.delete

6.quit

Enter ur choice:4

Enter the element after which to insert:12

The element is present

Enter the element to insert:10

The element is inserted

Singly Linked List

1.create

2.Display

3.search

4.insert

5.delete

6.quit

Enter ur choice:2

12->10->11->NULL

Singly Linked List

1.create

2.Display

3.search

4.insert

5.delete

6.quit

Enter ur choice:5

Enter the element u want to delete:10

The element is present

The element is deleted

Singly Linked List

1.create

2.Display

3.search

4.insert

5.delete

6.quit

Enter ur choice:2

12->11->NULL

Singly Linked List

1.create

2.Display

3.search

4.insert

5.delete

6.quit

Enter ur choice:6

**RESULT:**

Thus the singly linked list is implemented and insert, delete and search operations are performed And verified.

# DOUBLE ENDED QUEUE (DEQUEUE)

## AIM:

To write a C program to implement a double ended queue (dequeue) with all possible operations on it.

## ALGORITHM:

1. Initialize all variables and functions.
2. Read choices.
3. If queue is not full, add items at front or back. And increment top value by 1.
4. Else print "Queue is full".
5. If Queue is not empty, delete items at front or back. And decrement top value by 1.
6. Else print "Queue is empty".
7. Display the queue items.
8. End of the program.

****** Program for Double Ended Queue ******

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define size      5
struct dequeue{
int deq[size];
int front,rear;
}Q;
int Qfull()
{
if(Q.rear==size-1)
return 1;
else
return 0;
}
```

```
int Qempty()
{
if((Q.front>Q.rear) || (Q.front==-1 && Q.rear==-1))
return 1;
else
return 0;
}
int insert_rear(int item)
{
if(Q.front==-1 && Q.rear==-1)
Q.front++;
Q.deq[++Q.rear]=item;
return Q.rear;
}
int del_front()
{
int item;
if(Q.front==-1)
Q.front++;
item=Q.deq[Q.front];
Q.deq[Q.front]=-1;
Q.front++;
return item;
}
int insert_front(int item)
{
int i,j;
if(Q.front==-1)
Q.front++;
i=Q.front-1;
while(i>=0)
```

```
{
Q.deq[i+1]=Q.deq[i];
i--;
}
j=Q.rear;
while(j>=Q.front)
{
Q.deq[j+1]=Q.deq[j];
j--;
}
Q.rear++;
Q.deq[Q.front]=item;
return Q.front;
}
int del_rear()
{
int item;
item=Q.deq[Q.rear];
Q.deq[Q.rear]=-1;
Q.rear--;
return item;
}
void display()
{
int i;
printf("\n Queue is    ");
for(i=Q.front;i<=Q.rear;i++)
printf(" %d ",Q.deq[i]);
}
void main()
{int choice,i,item;
```

```
Q.front=-1;
Q.rear=-1;
for(i=0;i<size;i++)
Q.deq[i]=-1;
clrscr();
printf("\n\n\n\n Double ended queue OR Dequeue\n");
do
{printf("\n1.Insert front\n2.Insert rear\n3.Delete front\n4.Delete rear\n");
printf("5.Display\n6.Exit\n");
printf("\nEnter ur choice:");
scanf("%d",&choice);
switch(choice)
{case 1:if(Qfull())
printf("\nDequeue is  full");
else
{printf("\nEnter item to be inserted:");
scanf("%d",&item);
Q.front=insert_front(item);
}break;
case 2:if(Qfull())
printf("\nDequeue is  full");
else
{printf("\nEnter item to be inserted:");
scanf("%d",&item);
Q.rear=insert_rear(item);
}break;
case 3:if(Qempty())
printf("\nDequeue is  empty");
else
{
item=del_front();
```

```
printf("\nThe item deleted from queue is  %d",item);
}break;
case 4:if(Qempty())
printf("\nDequeue is  empty");
else
{
item=del_rear();
printf("\nThe item deleted from queue is  %d",item);
}
break;
case 5:display();
break;
case 6:exit(0);
}
}while(choice!=6);
getch();
}
```

OUTPUT:

1.Insert front

2.Insert rear

3.Delete front

4.Delete rear

5.Display

6.Exit

Enter ur choice:1

Enter item to be inserted:11

1.Insert front

2.Insert rear

3.Delete front

4.Delete rear

5.Display6.Exit

Enter ur choice:2

The item deleted from queue is 12

1.Insert front

2.Insert rear

3.Delete front

4.Delete rear

5.Display

6.Exit

Enter ur choice:5

Queue is 11 12

1.Insert front

2.Insert rear

3.Delete front

4.Delete rear

5.Display

6.Exit

Enter ur choice:1

Enter item to be inserted:10

1.Insert front

2.Insert rear

3.Delete front

4.Delete rear

5.Display

6.Exit

Enter ur choice:2

Enter item to be inserted:13

1.Insert front

2.Insert rear

3.Delete front

4.Delete rear

5.Display

6.Exit

Enter ur choice:5

Queue is 10 11 12 13

1.Insert front

2.Insert rear

3.Delete front

4.Delete rear

5.Display

6.Exit

Enter ur choice:3

The item deleted from queue is 10

1.Insert front

2.Insert rear

3.Delete front

4.Delete rear

5.Display

6.Exit

Enter ur choice:4

The item deleted from queue is 13

1.Insert front

2.Insert rear

3.Delete front

4.Delete rear

5.Display

6.Exit

Enter ur choice:5

Queue is 11 12

1.Insert front

2.Insert rear

3.Delete front

4.Delete rear

5.Display

6.Exit

Enter ur choice:6

**RESULT:**

Thus the C program for Double ended queue is implemented and insertion, deletion on both ends is done successfully.